Escola de Camins
Escola Tècnica Superior d'Enginyeria de Camins, Canals i Ports
UPC BARCELONATECH

TREBALL FINAL DE MÀSTER

X-FEM de alto orden para problemas bimateriales y aplicación al problema de Stefan

Treball realitzat per:
Alberto Peña Hernando

Dirigit per:
Sonia Fernández Méndez

Màster en:
Enginyeria de Camins, Canals i Ports

Barcelona, juny 2016

Departament d'Enginyeria Civil i Ambiental

Resumen

En este trabajo se usa el método de elementos finitos extendidos (X-FEM) para solucionar ecuaciones en derivadas parciales (EDPs) asociadas a una determinada geometría en la que existen discontinuidades. La gran ventaja de X-FEM es que no es necesario ajustar la malla a la geometría como en el método de elementos finitos estándar (MEF). El trabajo se divide en dos partes. En la primera se resuelve una EDP estacionaria en un dominio con un agujero y se explican los fundamentos del X-FEM. En la segunda parte se resuelve una EDP transitoria para un problema bimaterial.

En la primera parte del trabajo se explica con detalle los procesos de obtención de la forma débil, definición de una función level-set para identificar la interfaz, obtención del sistema a resolver, la imposición de las condiciones de contorno, la integración numérica en el dominio, teniendo en cuenta que existen elementos cortados por la interfaz, y la convergencia del método calculando el error cometido en la solución numérica (utilizando X-FEM) comparándola con la solución analítica del problema.

En la segunda parte se trata un problema bimaterial y transitorio. Tras aplicar el método de elementos finitos se obtiene un sistema de ecuaciones diferenciales ordinarias (EDOs). Se soluciona este sistema con el método de Euler, obteniendo la solución en cada paso de tiempo. Se resuelve este problema para una interfaz fija en el tiempo y para una interfaz móvil. Al trabajar con la interfaz móvil se debe actualizar la función level-set en cada paso de tiempo. Para ello, se tiene que calcular la velocidad normal en la interfaz y también se debe proyectar la solución en cada paso de tiempo ya que, al moverse la interfaz, los dominios de cada uno de los materiales, Ω_1 y Ω_2, no son constantes en el tiempo.

La EDP que se ha solucionado es la ecuación de Laplace. La interpretación física que se le ha dado en este trabajo es la distribución de temperaturas en un dominio. Pero aparte de esta interpretación, la ecuación de Laplace también modeliza problemas de consolidación, de flujo potencial, de flujo en medio poroso, etc. También se pueden solucionar problemas de fluidos con la ecuación de Navier-Stokes, cambiaría la forma débil del problema, pero las técnicas utilizadas serían las mismas. En la segunda parte del trabajo se soluciona el problema de Stefan: agua líquida y hielo separados por una interfaz móvil.

Abstract

In this paper the eXtended Finite Element Method (X-FEM) is used to solve partial differential equations (EDP's) associated with a geometry with discontinuities. The great advantage of X-FEM is that the computational mesh is not required to fit the interface, as happens in the standard Finite Element Method (FEM). This paper can be divided in two parts. The first one conveys a stationary PDE in a domain with a void that is solved in order to explain the basics of the X-FEM. In the second part a transient PDE for a bi-material problem is solved.

In the first part the fundamentals of the X-FEM are explained: obtaining the weak form of the problem, the definition of a level-set function in order to identify the interface, obtaining the system of equations to solve the problem, imposing boundary conditions, numerical integration on the domain, taking into account that there are some elements cut by the interface, and the convergence of the method calculating the error committed in the numerical solution (using X-FEM) by comparison with the analytical solution of the problem.

In the second part a transient and bi-material problem is solved. After applying X-FEM, a system of ordinary differential equations (ODE's) is obtained. The solution in each time step can be obtained solving this system of ODE's, using Euler method. Two different approaches have been made to solve this problem: fixed and moving interface. In order to work with a moving interface the level-set function must be update in each time step. To do that, the normal velocity on the interface must be calculated and a projection of the solution must be obtained because, as the interface moves in each time step, the domains of the materials, Ω_1 y Ω_2, are not constant in time.

The Laplace equation is solved in this paper. The physical interpretation given to the equation in this work is the distribution of the temperatures in a domain. But, there are more physical interpretations: consolidation problems, potential flow problems, porous medium flow problems,etc. Also, changing the weak form of the problem, the Navier-Stokes equation can be solved using these techniques. In the second part of the paper the Stefan problem is solved: liquid water and ice divided by a moving interface.

Índice

1. Introducción

El método de los elementos finitos, MEF, (FEM en inglés) es un método numérico que permite solucionar ecuaciones en derivadas parciales. Es un método muy potente con un gran campo de actuación en ingeniería. Las ecuaciones están asociadas a un problema físico sobre una determinada geometría. Sobre ésta, se crea una malla de tal manera que se divide en pequeños elementos. Cada uno de estos elementos tiene una serie de nodos; la solución del problema se obtiene primero en estos nodos y después se realiza una aproximación para todos los demás puntos del dominio. Cuánto más fina sea la malla y más nodos tengan los elementos mejor será esta aproximación. Sin embargo el coste computacional se incrementa.

Cuando en la geometría existen discontinuidades, agujeros o alguna otra interfaz se debe adaptar la malla a dicha interfaz. Por ejemplo, si en la geometría hay un agujero se debe mallar de forma que la interfaz coincida con los lados de los elementos; de tal manera que ningún elemento esté cortado por la interfaz. Este proceso no es eficiente y esta ineficiencia se multiplica en problemas transitorios en los que la interfaz se mueve, ya que se debe mallar continuamente con el coste computacional que esto supone.

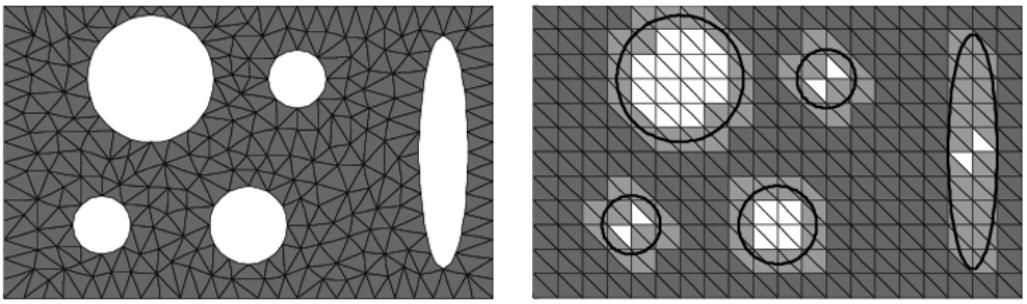

Figura 1: *Comparación mallas FEM y X-FEM*

Para solventar este problema se desarrolló un nuevo método, *eXtended Finite Element Method* , X-FEM, (método de elementos finitos extendidos). Este método permite no adaptar las mallas a la geometría cuando existe una discontinuidad o agujero. En la Figura 1, se puede ver la diferencias de mallado entre el MEF estándar y el X-FEM. Al realizar los cálculos pertinentes para solucionar el problema se debe tener en cuenta los distintos tipos de

elementos. En la Figura 1 los elementos estándar, no cortados por la interfaz, son los de color gris oscuro, los elementos cortados por la interfaz son gris claro y los blancos son los elementos del agujero, que no pertenecen al dominio.

En este trabajo se usa X-FEM para solucionar ecuaciones en derivadas parciales asociadas a una geometría en la que existe una interfaz. Se propone un problema estacionario y otro transitorio.

2. Estado actual del conocimiento y objetivos

El método de elementos finitos extendidos (X-FEM) se desarrolló hace más de una década para solventar problemas de ineficiencia en el FEM cuando existían discontinuidades en la geometría del problema, como ya se ha comentado en la introducción. Desde entonces, se ha trabajado para mejorarlo y solucionar los problemas que van surgiendo. Se ha aplicado a problemas con interfaces (bimateriales, con agujeros, etc) usando aproximaciones de alto orden, no sólo lineales. Para solucionar problemas de discontinuidades fuertes se realiza un enriquecimiento de la interpolación, usando, por ejemplo, una función de Heaviside. Una discontinuidad fuerte implica un cambio brusco de la solución al cambiar de dominio.

En este trabajo se soluciona el problema de Stefan, un problema bimaterial en el que se trata la interactuación de agua líquida con hielo y la evolución de la interfaz que separa ambas fases bajo unas determinadas condiciones de contorno. En este problema la condición en la interfaz es temperatura igual a temperatura de fusión para ambos dominios. El problema se podría resolver con enriquecimiento Heaviside, pero aquí se trata como dos problemas separados en agua y hielo. Hasta ahora se ha solucionado con una interpolación lineal de la solución, sin embargo, en este trabajo se realizará también para alto orden.

Personalmente, me familiaricé con el Método de Elementos Finitos en el primer curso del Máster de Caminos, Canales y Puertos con las asignaturas "*Modelización numérica*" e "*Ingeniería Computacional*". En ellas aprendí a obtener la forma débil de una ecuación en derivadas parciales, a discretizar el dominio, trabajar con funciones de forma, plantear el sistema de ecuaciones a resolver, aprendiendo a calcular la matriz de rigidez, la matriz de masa y

el vector de términos independientes elementales y cómo hacer el ensamblaje para obtener las matrices globales.

El objetivo de este proyecto es aprender a trabajar con el MEF, y especialmente con el X-FEM, de una manera más profunda con la ayuda de un software matemático (MatLab). El proyecto se puede dividir en dos partes. En la primera, el objetivo es familiarizarse con el método X-FEM resolviendo una ecuación en derivadas parciales definida en una determinada geometría. Esta geometría tiene un agujero. Por lo tanto, se trabajará con funciones $level - set$ para identificar qué elementos de la malla pertenecen al dominio y cuáles forman parte del agujero. Se aprenderá a trabajar con los elementos que han sido cortados por la interfaz y que sólo una parte de ellos pertenece al dominio. Se utilizará la integración numérica para solucionar las distintas integrales definidas en el dominio; para ello se trabajará con elementos de referencia y se deberá realizar el cambio de coordenadas oportuno. También se deberán imponer condiciones de contorno tanto en fronteras exteriores como en la interfaz, en este segundo caso se utilizará el método de Nitsche.

En la segunda parte del proyecto se tratará el problema de Stefan del que se ha hablado anteriormente. Al ser un problema transitorio, se debe estudiar la interactuación de las dos fases, agua y hielo, para ver cómo evoluciona el problema en el tiempo. Se estudiará el problema con una interfaz fija y también con interfaz móvil. En este último caso, se deberá actualizar la función level-set con el tiempo para mover la interfaz; así se conseguirá cambiar los dominios de cada una de las dos fases sin necesidad de cambiar la malla.

Para solucionar estos problemas se han creado varios códigos en MatLab. Algunas funciones utilizadas en este trabajo han sido realizados por otras personas. En la siguiente lista se citan las funciones disponibles inicialmente y se explica su función.

- createMesh: Función que crea una malla estructurada en un rectángulo. Devuelve la matriz con las coordenadas de los nodos, X, y la matriz de conectividad, T.
- createReferenceElement: Función que crea el elemento de referencia, incluyendo las coordenadas de los nodos, puntos de integración, pesos y valor de las funciones de forma y sus derivadas en los puntos de integración para el elemento $1D$ y $2D$.

- computeSystemLaplace: Función que calcula la matriz K y el vector f de una ecuación de Laplace para FEM.
- computeL2Norm: Función que calcula el error en norma L2 comparando una solución de FEM con una solución analítica.
- ModifyQuadrature: Si un elemento está cortado por una interfaz esta función devuelve una lista de puntos que definen dicha interfaz, además de puntos y pesos de integración del elemento cortado.
- computeShapeFunctions2D: Función que, al darle las coordenadas de ciertos puntos, devuelve el valor de las funciones de forma y sus derivadas en dichos puntos.
- computeL2projection2levelset, computeLeastSquaresMatricesCutElements: El objetivo de estas dos funciones es realizar la proyección L2 de la solución para tener en cuenta el cambio de dominio con una interfaz móvil.

A partir de estos códigos de FEM se crearon los códigos de XFEM. La mayor diferencia consiste en tratar con elementos cortados por la interfaz. Así, se modificaron las funciones anteriores para tener esto en cuenta. Los códigos realizados por el autor se explican con detalle en los anejos al final de la memoria. Son los siguientes.

- MainXFEM: Programa principal que, llamando a otras funciones, da como resultado la solución de la ecuación estacionaria en derivadas parciales.
- MainTransientNotMovingInterface: Programa que soluciona una ecuación transitoria en derivadas parciales con una interfaz fija.
- MainTransientMovingInterface: Programa que soluciona el problema de Stefan con una interfaz móvil.
- computeSystemLaplaceCutElements: Función que calcula la matriz K y el vector f para elementos cortados.
- computeSystemLaplaceCutElementsNitsche: Función que modifica la matriz K y el vector f para tener en cuenta los términos que se añaden en el método de Nitsche.

- computeSystemLaplaceTransient: Función que calcula la matriz K y la matriz de masa, M, en un problema transitorio. Para elementos no cortados.
- computeSystemLaplaceCutElementsTransient: Función la K y la M para elementos cortados.
- computeL2NormCut: Función que calcula el error utilizando la norma L2 en elementos cortados.
- calculaVelocidadNormalInterfaz: Función que calcula la velocidad normal en la interfaz a partir del salto de flujos de la solución. Es necesario para actualizar el level set en el tiempo.

3. Problema estacionario

3.1. Planteamiento del problema

Se plantea la siguiente ecuación en derivadas parciales de Laplace con condiciones de contorno tipo Dirichlet.

$$\begin{cases} -\bar{\nabla} \cdot (\nu \bar{\nabla} u) = f \\ u = u_D \quad en \quad \Gamma_D \\ u = u_I \quad en \quad I \end{cases} \tag{1}$$

Una interpretación física del problema es la distribución de temperaturas en el dominio Ω. Se fija una temperatura en los contornos, u_D, y la temperatura fluye hasta alcanzar un estado estacionario. La difusión, ν, depende de las propiedades del material. También hay que tener en cuenta el término fuente que introduce calor (puede subir o bajar la temperatura) en el dominio. Una vez se haya llegado al estado estacionario, la distribución de temperaturas será la solución del problema, u.

Esta ecuación también modeliza un problema de consolidación, de flujo potencial, de flujo en medio poroso, etc.

3.1.1. Dominio del problema

En el programa implementado en MatLab la EDP se plantea para un dominio general Ω. En este ejemplo, se define Ω como un cuadrado con vértices

en $(-1,-1)$, $(1,-1)$, $(1,1)$ y $(-1,1)$ y con un agujero en el medio. Este agujero es una círculo con origen en el punto $(0,0)$ y de radio 0,5. Por lo tanto el dominio tiene frontera exterior e interior. La frontera exterior son los lados del cuadrado, se le denomina Γ_D porque en ese contorno se han impuesto condiciones tipo Dirichlet. A la frontera interior se le denomina interfaz, I, está definida con una función level-set.

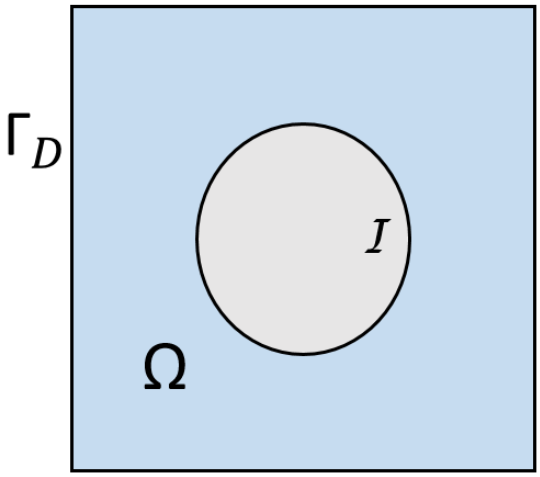

Figura 2: *Dominio del problema.*

3.1.2. Función Level-Set

Una función Level-Set se usa cuando existen distintos dominios en la geometría. Por ejemplo, un problema con la misma geometría que el definido anteriormente pero dos materiales con propiedades distintas (en vez de un material en un dominio y un agujero en el medio). Así, se tiene un dominio $\Omega = \Omega_1 \cup \Omega_2$, diferenciando los dos tipos de materiales. La interfaz, I, representa este cambio de dominio. Para cada punto, $\mathbf{x}$ la función Level-Set, $\varphi(\mathbf{x})$, identifica en qué dominio se encuentra dicho punto.

$$\begin{cases} \varphi(\mathbf{x}) > 0 & \mathbf{x} \in \Omega_1 \\ \varphi(\mathbf{x}) < 0 & \mathbf{x} \in \Omega_2 \\ \varphi(\mathbf{x}) = 0 & \mathbf{x} \in I \end{cases} \tag{2}$$

En el problema que se trata en este apartado con la función level-set se identifica si un punto $\mathbf{x}$ pertenece al dominio, Ω_1, a la interfaz, I, o forma parte del agujero, Ω_2, y por tanto no pertenece al dominio del problema.

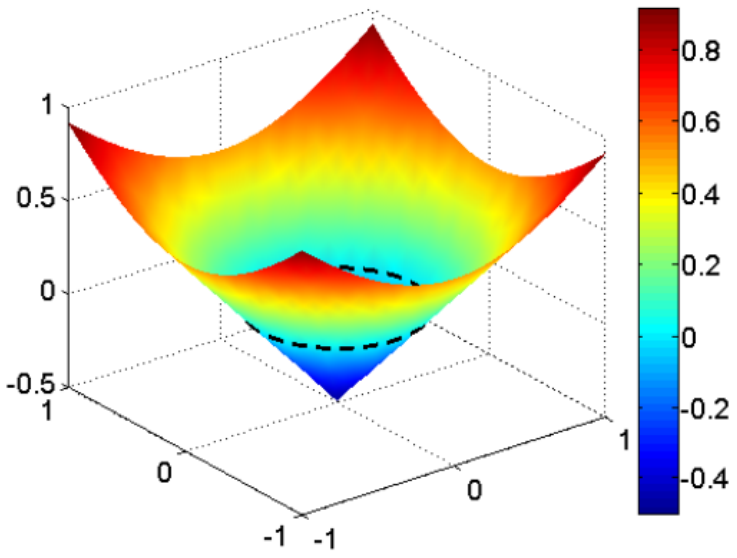

Figura 3: *Gráfica en 3D de una función Level-Set.*

3.1.3. Mallado

Como se ha explicado en la introducción la ventaja de X-FEM es poder trabajar con mallas no adaptadas a la interfaz, en este caso regulares. En este trabajo se emplean elementos de tipo triangular. El tamaño de los elementos (o tamaño de malla, h) puede variar; con una malla más fina (un tamaño de elemento menor) los resultados serán más precisos pero también el coste computacional del programa será mayor. El número de nodos en el elemento también varía; con una interpolación lineal se obtienen triángulos de 3 nodos, uno en cada vértice. Con una interpolación de grado 2, por ejemplo, se obtienen triángulos con 6 nodos.

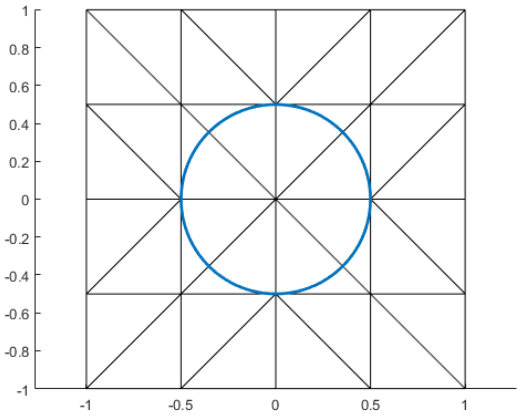

Figura 4: *Imagen de la malla con $h = 0.5$.*

Existen tres tipos de elementos; elementos estándar, elementos pertenecientes al agujero y elementos cortados. Los elementos estándar pertenecen al dominio Ω, los pertenecientes al agujero no. Los elementos cortados tienen parte que pertenece al dominio y otra parte que no. Más adelante, en la sección 3.5 se explica con detalle cómo trabajar con los diferentes tipos de elementos.

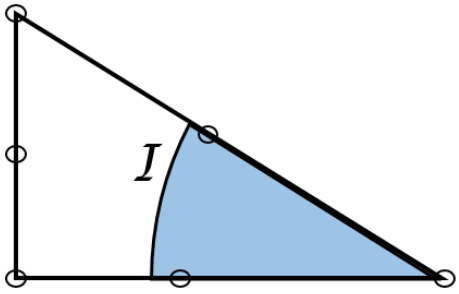

Figura 5: *Ejemplo de elemento de grado 2 (6 nodos) cortado por la interfaz*

3.2. Obtención de la forma débil y método de Nitsche

Para solucionar el problema planteado en (1) el primer paso es encontrar la forma débil de éste. Para ello, primero se premultiplica por v a ambos lados de la ecuación y se integra en el dominio; v es una función test tal que

$v = 0$ en Γ_D.

$$\int_\Omega v(-\bar{\nabla} \cdot (\nu \bar{\nabla} u)\, d\Omega = \int_\Omega vf\, d\Omega \tag{3}$$

Se toma la difusión, ν, constante; entonces se puede sacar fuera del operador $\bar{\nabla}$. Integrando por partes y reordenando los términos queda de la siguiente manera

$$\nu \int_\Omega (\bar{\nabla} v)\bar{\nabla} u\, d\Omega - \nu \int_{\Gamma_D} v(\bar{\nabla} u \cdot \mathbf{n})\, d\Gamma - \nu \int_I v(\bar{\nabla} u \cdot \mathbf{n})\, dl = \int_\Omega vf\, d\Omega. \tag{4}$$

El segundo y el tercer término resultan de la integración por partes. El segundo se puede eliminar ya que la función test v es nula en el contorno de Dirichlet.

La imposición de las condiciones de contorno en la frontera exterior se trata en la sección 3.4. Imponer las condiciones de contorno en la interfaz, I, no es tan trivial como en la frontera exterior. Se trata en este apartado ya que se imponen modificando la forma débil del problema.

La interfaz I corta a diversos elementos de la malla creando en cada elemento una parte perteneciente al dominio Ω y otra perteneciente al agujero. Las condiciones de contorno se deben imponer en la interfaz I, de tal forma que, al contrario que en el caso de la frontera exterior, no hay una serie de nodos de elementos a lo largo de la frontera a los que asignar el valor prescrito. Se debe imponer las condiciones de contorno en ciertos puntos de la interfaz. Además, las funciones de forma no cumplen la propiedad de la delta de Kronecker en estos puntos como sí lo hacen en los nodos de los elementos. Para solventar este problema se utiliza el método de Nitsche, originalmente utilizado para los métodos que no utilizan malla (*"Mesh-free methods"*).

Existen otros métodos que modifican la forma débil como el método de los multiplicadores de Lagrange y el método del penalty. El inconveniente del método de Nitsche con respecto a los otros dos es que la modificación de la forma débil es menos trivial; ya que para cada problema es diferente. La gran ventaja es que no sufre de mal condicionamiento; es mucho más estable. Este método depende de un parámetro β. Se puede asegurar la convergencia a la solución si se coge $\beta = \alpha/h$ donde α es una constante lo suficientemente grande y h es el tamaño de malla. Por tanto, cuánto más fina sea la malla más grande será el parámetro β.

La forma débil del problema (1) es: Encontrar u que cumpla $u = u_D$ en Γ_D y la ecuación

$$\begin{aligned}\nu \int_\Omega (\bar{\nabla} v)\bar{\nabla} u \, d\Omega - \nu \int_I \mathbf{v}(\bar{\nabla} u \cdot \mathbf{n}) \, dl - \nu \int_I (\bar{\nabla} v \cdot \mathbf{n}) u \, dl + \beta\nu \int_I vu \, dl = \\ = \int_\Omega vf \, d\Omega - \nu \int_I u_I (\bar{\nabla} v \cdot \mathbf{n}) \, dl + \beta\nu \int_I u_I v \, dl\end{aligned} \tag{5}$$

para todo v, tal que $v = 0$ en Γ_D.

Como se puede observar, las condiciones de contorno en la frontera exterior se imponen. Sin embargo, las condiciones de contorno en la interfaz se tienen en cuenta añadiendo términos a la ecuación. u_I es el valor de la condición de contorno en la interfaz.

En (5) el término $\int_I v(\bar{\nabla} u \cdot \mathbf{n}) \, dl$ se obtiene al integrar por partes $\bar{\nabla} \cdot (\bar{\nabla} u)$, el término $\int_I (\bar{\nabla} v \cdot \mathbf{n}) u \, dl$ recobra la simetría del problema y el término $\beta \int_I vu \, d\Gamma$ asegura la coercitividad. Los nuevos términos en el lado derecho de la ecuación son para asegurar la consistencia del problema.

En la ecuación (5) se deben solucionar integrales de línea a lo largo de la interfaz I. En la sección 3.5 se explica con detalle cómo realizar las distintas integrales planteadas.

3.3. Obtención del sistema

Para solucionar la ecuación (5) se considera una interpolación seccional de la solución

$$u(x) \approx u^{(h)}(x) = \sum_i u_i N_i(x) = [N_1, ..., N_n] \begin{bmatrix} u_1 \\ \dots \\ \dots \\ \dots \\ u_n \end{bmatrix} = \mathbf{N}\,\mathbf{u}, \tag{6}$$

donde $\mathbf{N}$ es la matriz de las funciones de forma. Estas funciones de forma cumplen la propiedad de la delta de Kronecker, $N_i(x_{ij}) = \delta_{ij}$. Así, en el nodo 5 por ejemplo, la función de forma correspondiente a ese nodo tomará valor 1, $N_5(x_5) = 1$, y todas las demás funciones de forma valdrán 0. El vector

solución **u** contiene la solución en los nodos, para todos los demás puntos de la geometría la solución se interpola con las funciones de forma.

Se considera también la matriz **G**, que contiene la información de las derivadas de dichas funciones

$$\bar{\nabla} u \approx \begin{bmatrix} \dfrac{\partial N_1}{\partial x} & \cdots & \dfrac{\partial N_n}{\partial x} \\ \\ \dfrac{\partial N_1}{\partial y} & \cdots & \dfrac{\partial N_n}{\partial y} \end{bmatrix} \begin{bmatrix} u_1 \\ \cdots \\ \cdots \\ \cdots \\ u_n \end{bmatrix} = \mathbf{G}\ \mathbf{u}. \tag{7}$$

Como (5) se debe cumplir para todo v, se toma $v = \mathbf{N}\mathbf{v}$ siendo $\mathbf{v}$ los valores nodales de v. Análogamente a lo anterior, $\bar{\nabla} v = \mathbf{G}\ \mathbf{v}$. Por tanto, se puede reescribir (5) de la siguiente manera

$$\mathbf{v}^T \left[\nu \int_\Omega \mathbf{G}^T \mathbf{G}\ d\Omega \right] \mathbf{u} = \mathbf{v}^T \int_\Omega \mathbf{N}^T f\ d\Omega. \tag{8}$$

Como (8) se debe cumplir para cualquier valor de $\mathbf{v}$, se pueden suprimir a ambos lados de la ecuación. Se obtiene un sistema $\mathbf{K}\mathbf{u} = \mathbf{f}$; donde $\mathbf{K}$ es la matriz de rigidez y $\mathbf{f}$ es el vector de términos independientes.

$$\mathbf{K} = \nu \int_\Omega \mathbf{G}^T \mathbf{G}\ d\Omega - \nu \int_I \mathbf{N}^T (\mathbf{n} \cdot \mathbf{G})\ dl - \nu \int_I (\mathbf{G}^T \cdot \mathbf{n}^T) \mathbf{N}\ dl + \beta\nu \int_I \mathbf{N}^T \mathbf{N}\ dl \tag{9}$$

$$\mathbf{f} = \int_\Omega \mathbf{N}^T f\ d\Omega - u_I \nu \int_I (\mathbf{G}^T \cdot \mathbf{n}^T)\ dl - u_I \beta\nu \int_I \mathbf{N}^T\ dl \tag{10}$$

3.4. Condiciones en el contorno exterior

Se imponen condiciones de contorno tipo Dirichlet, ($\bar{u} = \bar{u}_D$). Se impone este tipo de condición de contorno tanto en la frontera exterior como en la interfaz. En el apartado 3.2 se han impuesto las condiciones de contorno en la interfaz mediante el método de Nitsche. En el contorno exterior es más sencillo ya que se puede asignar el valor prescrito, $\bar{u}_D$, directamente a los nodos de los elementos que formen parte de dicha frontera. Por tanto, una vez se hayan obtenido la matriz $\mathbf{K}$ y el vector $\mathbf{f}$ se debe hacer una reducción del sistema; se eliminan las filas y columnas de la matriz y el vector correspondientes a los nodos con valor ya asignado por las condiciones de contorno.

Se soluciona el sistema $\mathbf{K}_R\ \mathbf{u}_R = \mathbf{f}_R$ con las matrices reducidas y se obtiene el vector reducido $\mathbf{u}_R$, solución de todos los nodos que no forman parte del contorno. Para obtener la solución completa, $\mathbf{u}$, se junta la información obtenida resolviendo el sistema con la dada en las condiciones de contorno.

3.5. Integración numérica

El objetivo de la integración numérica es calcular (aproximar) el valor de una integral. Se usa cuando $f(x)$ tiene una expresión analítica desconocida (datos experimentales o función discreta) o la integral analítica es desconocida o muy complicada.

$$I = \int_a^b F(x)\,dx \approx \sum_{i=1}^{n} \omega_i F(x_i) \tag{11}$$

Se aproxima la integral mediante un sumatorio, en el caso 1D se evalúa la función en n puntos (x_i) ponderados por un peso (ω_i).

Existen varias técnicas para obtener estos puntos; Newton-Cotes, generalmente equiespaciados, Gauss, puntos óptimos para que la cuadratura sea lo mejor posible y también existen técnicas mixtas: Radau, Lobeto. En el MEF generalmente se utiliza la cuadratura de Gauss.

3.5.1. Integración en elementos estándar

Un elemento estándar es aquel que no está cortado por la interfaz. Por lo tanto, se debe integrar en todo el elemento. Para realizar esta integración se utiliza el elemento de referencia. El elemento de referencia en este caso es un triángulo de vértices $(0,0),(0,1)$ y $(1,0)$, definido en el sistema de coordenadas (ξ,η). Para cambiar de sistema de coordenadas se realiza una transformación isoparamétrica. Se define $\bar{x} = \sum_i N_i(\xi)\bar{x}_i$, así

$$\frac{\partial F_j}{\partial \bar{x}_i} = J^{-1} \cdot \frac{\partial F_j}{\partial \bar{\xi}_i} \tag{12}$$

donde J es la matriz Jacobiana de la transformación

$$J = \begin{bmatrix} \dfrac{\partial x}{\partial \xi} & \dfrac{\partial y}{\partial \xi} \\ \\ \dfrac{\partial x}{\partial \eta} & \dfrac{\partial y}{\partial \eta} \end{bmatrix}. \tag{13}$$

Por lo tanto, la integral que se debe resolver en el elemento físico se transforma en una integral en el elemento de referencia.

$$\int_{\Omega_e} F(\bar{x})\, d\bar{x} = \int_{\Omega_{ref}} F(\bar{x}(\bar{\xi}))|J(\bar{\xi})|\, d\bar{\xi} \tag{14}$$

En el elemento de referencia sí se dispone de información de los puntos y pesos de integración, por lo tanto se puede realizar la integración numérica.

$$\int_{\Omega_{ref}} F(\bar{x}(\bar{\xi}))|J(\bar{\xi})|\, d\bar{\xi} = \sum_{i=1}^{n} \omega_i F(\bar{x}(\bar{\xi}_i))|J(\bar{\xi}_i)| \tag{15}$$

3.5.2. Integración en elementos cortados

Se ha definido la interfaz I como los puntos de valor 0 de una función *level set* φ. Así $I = \{x \mid \varphi(x) = 0\}$. Por lo tanto, dicha interfaz cortará a varios elementos de la malla definida en el dominio. Estos elementos cortados están divididos en dos subregiones, una correspondiente al dominio del material y la otra al agujero. La cuadratura numérica se debe definir sólo en el dominio del material.

Se ha de representar correctamente la interfaz en los elementos cortados. Una representación lineal de la interfaz es válida para aproximaciones lineales pero no para aproximaciones de mayor orden ya que la precisión es limitada. En X-FEM típicamente se usan dos métodos: *Oct-tree partition* y *K-th degree parametrization* para aproximar la interfaz. En el primer método se hace una aproximación lineal a trozos de la interfaz en cada elemento cortado y en el segundo, el utilizado en este trabajo, se realiza una parametrización de la interfaz de grado k.

Una vez se ha representado la interfaz se deben obtener puntos de integración en cada una de las dos subregiones. En nuestro caso se realizará la cuadratura numérica únicamente con los puntos de integración de la subregión correspondiente al dominio del material, procediendo igual que en el apartado anterior.

3.5.3. Integración numérica en la interfaz

Para la imposición de condiciones de contorno Dirichlet en la interfaz, se ha modificado la forma débil del problema mediante el método de Nitsche. En este método, se han de realizar integrales en la interfaz I.

Se realiza una parametrización de la curva (de la interfaz) a partir de los valores de las funciones de forma en el elemento de referencia $1D$. Se debe diferenciar el elemento de referencia $1D$, el elemento de referencia $2D$ y el elemento físico.

El elemento de referencia $1D$ es un segmento que va de *-1* a *1*. En él se conoce el valor de las funciones de forma y los puntos y pesos de integración. Es la representación en $1D$ de la Interfaz.

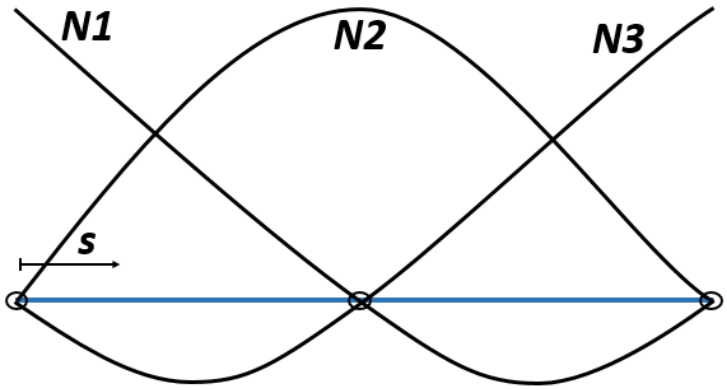

Figura 6: *Elemento de referencia 1D y funciones de forma para grado 2*

El elemento de referencia $2D$ es el comentado anteriormente. Triángulo de vértices $(0,0),(0,1)$ y $(1,0)$, definido en el sistema de coordenadas (ξ,η).

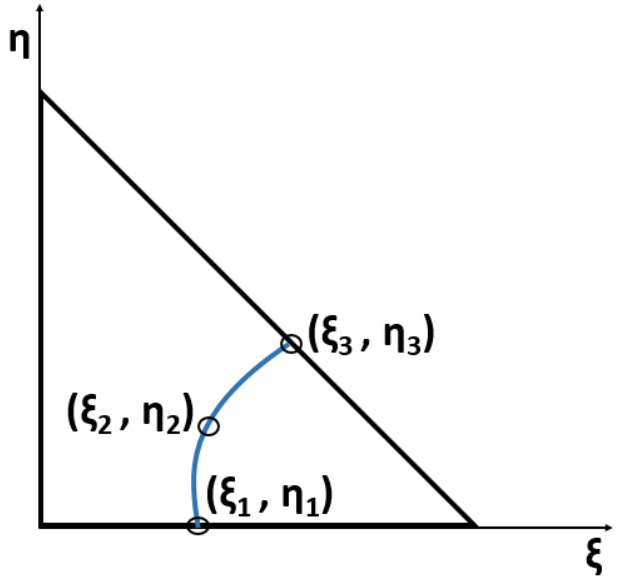

Figura 7: *Elemento de referencia 2D*

En él se representa la curva (interfaz I), parametrizada de la siguiente manera.

$$\begin{cases} \xi(s) = \sum_{i=1}^{n} \xi_i N_i^{1D}(s) \\ \eta(s) = \sum_{i=1}^{n} \eta_i N_i^{1D}(s) \end{cases} \tag{16}$$

El elemento físico es cada uno de los triángulos definidos al mallar. Se encuentran en el sistema de coordenadas (x, y). La curva I_e es la intersección de la interfaz con un elemento físico, $I_e = I \cap \Omega_e$.

La curva se parametriza de igual forma que en el elemento de referencia.

$$\begin{cases} x(s) = \sum_{i=1}^{n} x_i N_i^{1D}(s) \\ y(s) = \sum_{i=1}^{n} y_i N_i^{1D}(s) \end{cases} \tag{17}$$

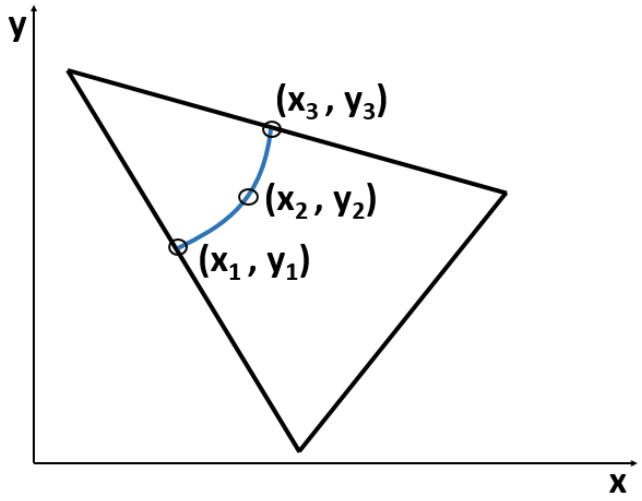

Figura 8: *Elemento físico cortado por la interfaz*

Los puntos $(\xi_i,\ \eta_i)$ y $(x_i,\ y_i)$ en las parametrizaciones de las curvas son los nodos de la curva I_e en el elemento de referencia y en el elemento físico, respectivamente . Se conoce la posición de estos nodos en los elementos de referencia $1D$ y $2D$. Se puede obtener la posición de estos nodos en el elemento físico mediante una transformación isoparamétrica:

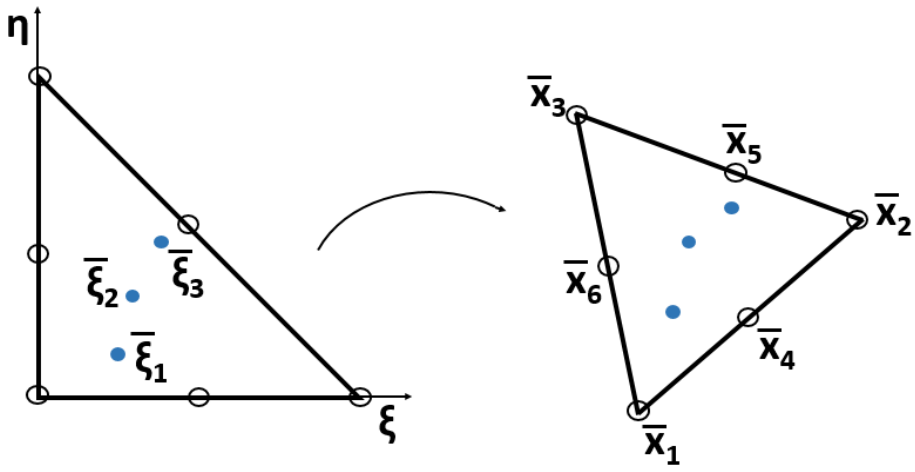

Figura 9: *Transformación Isoparamétrica*

$$\bar{x_j} = \sum_i \bar{x}_i N_i^{2D}(\bar{\xi_j}). \tag{18}$$

Como se ve en (18) esta transformación sirve para cualquier punto, y en particular, para los nodos de la curva. Para obtener las coordenadas de un punto $\bar{x}_j$ en el elemento físico se necesitan las coordenadas de los nodos del elemento físico y el valor de las funciones de forma 2D en el punto correspondiente en el elemento de referencia $\bar{\xi}_j$.

El objetivo es calcular la integral de una función $F(\bar{x})$a lo largo de la interfaz I. Se puede descomponer en un sumatorio a lo largo de los elementos cortados

$$\int_I F(\bar{x})dl = \sum_{\Omega_e cortados} \int_{I_e} F(\bar{x})dl. \tag{19}$$

Para realizar cada una de estas integrales en I_e se transforma a una integral en el elemento de referencia $1D$. Por lo tanto será una integral de *-1* a *1*. Para el cambio de coordenadas, en $2D$ se utilizaba el Jacobiano, ahora en $1D$ se utiliza el módulo de la derivada $\|\dot{\bar{x}}(s)\|$. Una vez la integral está definida en el elemento de referencia se puede resolver mediante una cuadratura numérica.

$$\int_{I_e} F(\bar{x})dl = \int_{-1}^{1} F(\bar{x}(s))\|\dot{\bar{x}}(s)\|ds \approx \sum_g F(\bar{x}(s_g))\|\dot{\bar{x}}(s_g)\|\omega_g \tag{20}$$

En (20) $\bar{x}(s_g)$ hace referencia a cada uno de los puntos de Gauss en el elemento físico. Para calcularlos se utiliza la parametrización de la curva definida en (17). $\bar{x}(s_g) = \sum_i \bar{x_i} N^{1D}(s_g)$. Siendo s_g los puntos de Gauss en el elemento de referencia 1D. Es necesario calcular el valor de la función F en estos puntos. Los pesos ω_g son conocidos. Y la derivada se calcula a partir de la derivada de las funciones de forma.

$$\dot{\bar{x}}(s_g) = \sum_i \bar{x}_i \frac{\partial N_i^{1D}(s_g)}{\partial s} \tag{21}$$

Una vez obtenida la derivada, se debe obtener la norma del vector y ya se tienen todos los elementos para realizar la integración numérica en la interfaz.

3.6. Convergencia y validación del método

En este apartado se estudiarán los resultados obtenidos, si son o no los esperados con las condiciones dadas, y se estudiará la convergencia del método; la solución debe ser cada vez mejor a medida que se refina la malla y se usa un grado mayor para interpolar la solución. Para estudiar la convergencia, lo más fácil es comparar la solución obtenida con el método numérico con una solución analítica ya conocida, así se puede calcular el error de la solución numérica. Sea $u_a = g(\mathbf{x})$ la solución analítica del problema (1) y u_{FEM} la solución numérica obtenida por el método de elementos finitos, la norma $L2$ (norma euclídea) se define de la siguiente manera

$$L2 = \sqrt{\int_\Omega u_{FEM} - u_a}. \tag{22}$$

Esta integral se calcula aproximando con una cuadratura de Gauss, como ya se ha visto anteriormente.

Para que la solución obtenida numéricamente sea lógica se debe tener en cuenta dos factores. Primero, si $g(\mathbf{x})$ es la solución analítica ha de cumplir (1), por lo tanto el término fuente se debe tomar consecuentemente. Por ejemplo, sea $g(\mathbf{x}) = x^2$; el término fuente, f, debe cumplir $f = -\bar{\nabla}\cdot(\nu\bar{\nabla}x^2)$, con lo que queda que $f = -2\nu$. La otra cosa que se debe tener en cuenta es la elección de los valores de las condiciones de contorno. Para que la solución tenga sentido, tanto los nodos del contorno exterior como los de la interfaz deben cumplir la función $g(\mathbf{x})$. En el ejemplo anterior, $g(\mathbf{x}) = x^2$, para cada nodo del contorno exterior se toma su coordenada x y se eleva al cuadrado, ésa será la condición de contorno. Para los nodos de la interfaz igual, simplemente se fija su valor con el método de Nitsche.

Para estudiar la convergencia, se deben obtener varias soluciones con distintas mallas y distintos grados de interpolación de la solución. Así, se sabe que el error cometido, E, sigue la siguiente relación: $E \sim Ch^{p+1}$. Siendo h el tamaño de elemento de la malla y p el grado de interpolación de la solución, se observa cómo el error se incrementa con el tamaño de malla; como es obvio, mallas más finas proporcionarán mejores resultados. Si en la anterior relación se toman logaritmos, se observa que se pueden relacionar el logaritmo del Error y el logaritmo del tamaño de malla mediante una recta de pendiente $(p + 1)$.

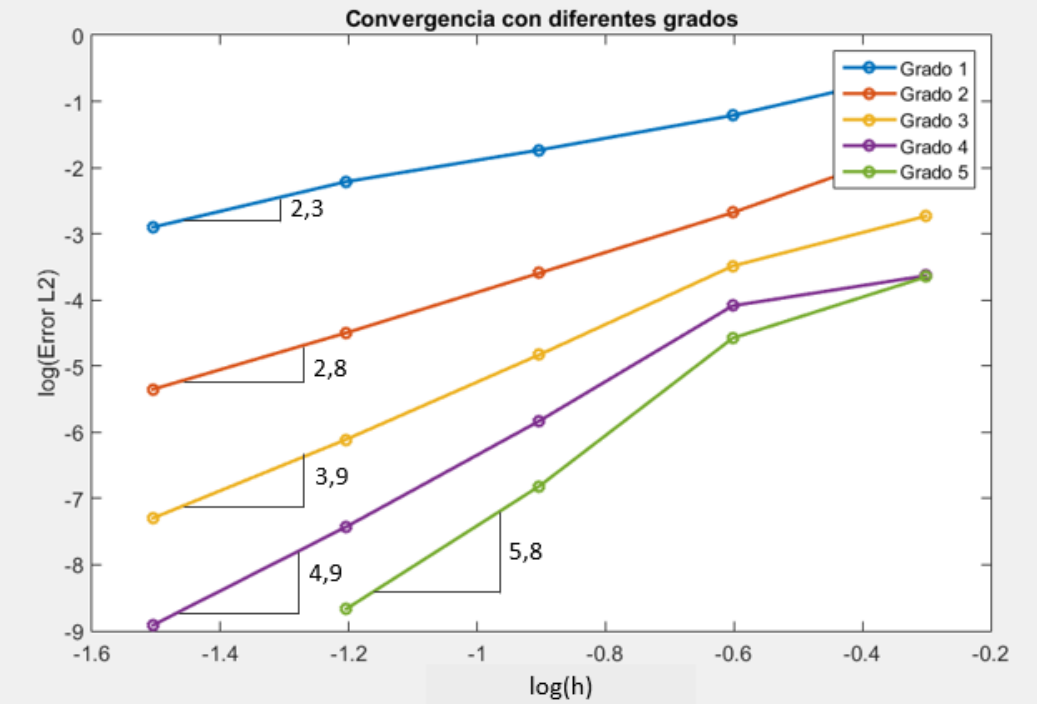

Figura 10: *Convergencia del método con distintos grados*

La Figura 10 muestra la convergencia para la solución analítica $u_a = g(\mathbf{x}) = sin(x + y)$ y y las condiciones de contorno acordes a dicha solución analítica. En la solución de grado 1 se tiene una recta de pendiente $m_1 \approx 2$, con grado 2 una recta de pendiente $m_2 \approx 3$, etc. Por lo tanto la convergencia del problema se comporta de manera esperada, con unos errores razonables y por tanto, se puede afirmar que la solución numérica es correcta.

La Figura 11 muestra la solución numérica del problema anterior. Los valores obtenidos son los esperados; se observa una simetría diagonal, se obtienen valores positivos cuando la suma de las coordenadas x+y es mayor que 0, valores negativos cuando la suma de x+y es menor que 0, el máximo (1) se obtiene en la zona en la que $x + y \approx 1{,}57 = \Pi/2$ y el mínimo (-1) en la zona en la que $x + y \approx -\Pi/2$. A la vista de los resultados, se puede concluir que el método propuesto es consistente.

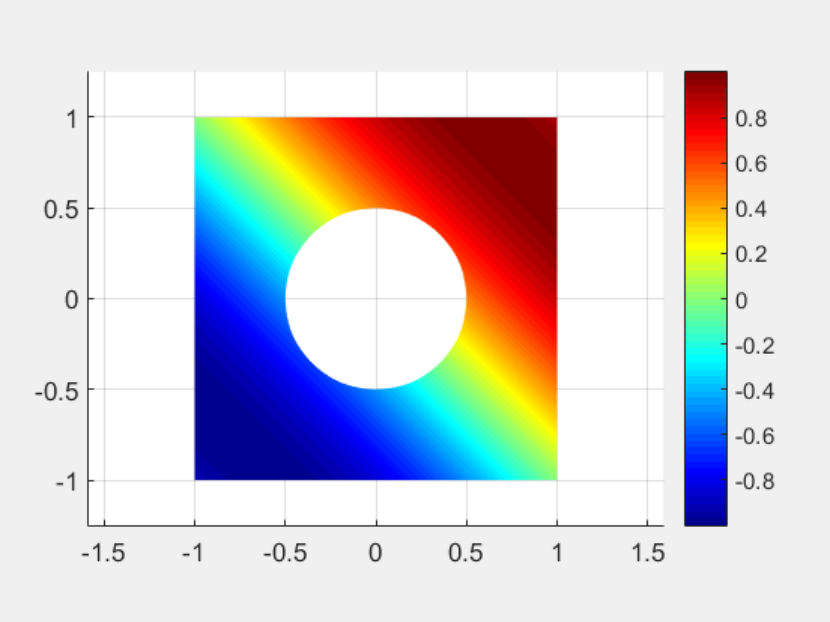

Figura 11: *Solución numérica con* $g = sin(x + y)$

4. Problema transitorio de dos fases

En este apartado se plantea el problema de Stefan; un problema bimaterial, agua líquida y hielo, y la interfaz que separa ambos materiales. Las ecuaciones que gobiernan dicho problema son las siguientes.

$$\begin{cases} c_1 \dfrac{\partial \mathbf{u}}{\partial t} - \bar{\nabla} \cdot (\kappa_1 \bar{\nabla} \mathbf{u}) = 0 \quad en \quad \Omega_1 \\ u = u_D \quad en \quad \Gamma_D \\ u = T_m \quad en \quad I \end{cases} \tag{23}$$

$$\begin{cases} c_2 \dfrac{\partial \mathbf{u}}{\partial t} - \bar{\nabla} \cdot (\kappa_2 \bar{\nabla} \mathbf{u}) = 0 \quad en \quad \Omega_2 \\ u = T_m \quad en \quad I \end{cases} \tag{24}$$

La primera ecuación gobierna el comportamiento del líquido y la segunda el del hielo. En ambos casos se fija la temperatura T_m en la interfaz. T_m es la temperatura de fusión, 0^oC en el caso del agua. La interfaz se comporta como la frontera entre el líquido y el sólido. En el caso del líquido también se fijan condiciones de contorno en la frontera exterior. Las constantes c_1 y

c_2 son las capacidades caloríficas volumétricas del agua líquida y del hielo respectivamente y las constantes κ_1 y κ_2 son las conductividades térmicas. $c1 = 0{,}62\ cal/^oC\ cm^3$, $c2 = 0{,}49\ cal/^oC\ cm^3$, $k1 = 6{,}9e-3\ cal/cm\ s\ ^oC$ y $k2 = 9{,}6e-3\ cal/cm\ s\ ^oC$.

La geometría será la misma que en el anterior ejemplo, pero en vez de tener un solo dominio Ω con un agujero en medio se tienen dos subdominios Ω_1 y Ω_2. Ω_1, la parte que en el anterior ejemplo era Ω, corresponde al líquido y Ω_2, lo que en el anterior ejemplo era el agujero, al sólido. Este problema bimaterial se podría solucionar con enriquecimiento Heaviside. Sin embargo, en este trabajo se ha optado por solucionarlo como dos problemas con agujero separados. Así, se tendrán dos problemas como el anterior.

La diferencia con el ejemplo anterior es que en las ecuaciones (23) y (24) existe un término transitorio; la solución $\mathbf{u}$ depende del tiempo. Para solucionar este problema se va a transformar las ecuaciones en derivadas parciales en un sistema de ecuaciones diferenciales ordinarias (EDOs) de primer orden.

Como las ecuaciones (23) y (24) son iguales pero cambiando de dominio y de propiedades del material, se trabajará con un dominio genérico Ω_i y unas propiedades c_i y κ_i; después sólo habrá que especificar en cada caso cuál es el dominio y las propiedades del fluido y del sólido respectivamente. Se define la difusión $\nu = \frac{\kappa}{c}$, por lo tanto se pueden reescribir las ecuaciones (23) y (24) de la siguiente manera.

$$\begin{cases} \dfrac{\partial \mathbf{u}}{\partial t} - \bar{\nabla} \cdot (\nu_i \bar{\nabla} \mathbf{u}) = 0 \quad en \quad \Omega_i \quad con \quad i = 1, 2 \\ u = T_m \quad en \quad I \\ u = u_D \quad en \quad \Gamma_D \end{cases} \tag{25}$$

El primer paso para lograr convertir la ecuación en un sistema de EDOs es encontrar la forma débil del problema, para ello se debe trabajar de la misma manera que en la sección 3.2. Las novedades frente al anterior problema son la existencia de un término transitorio, derivada de la solución $\mathbf{u}$ con respecto al tiempo, y la ausencia de término fuente.

Premultiplicando por $\mathbf{v}$ ($\mathbf{v} = 0$ en Γ_D) e integrando en el dominio Ω_i, se obtiene lo siguiente

$$\int_{\Omega_i} \mathbf{v}\frac{\partial \mathbf{u}}{\partial t}\, d\Omega_i - \int_{\Omega_i} \mathbf{v}(\bar{\nabla}\cdot(\nu_i\bar{\nabla}\mathbf{u})\, d\Omega = 0. \tag{26}$$

Integrando por partes el segundo término, como en la sección 3.2 y aplicando el método Nitsche para imponer condiciones de contorno en la frontera se puede reescribir (25) de la siguiente manera:

Encontrar u que cumpla $u = u_D$ en Γ_D y la ecuación

$$\int_{\Omega_i} v\frac{\partial u}{\partial t}\, d\Omega_i + \nu_i \int_{\Omega_i} (\bar{\nabla}v)\bar{\nabla}u\, d\Omega_i - \nu_i \int_I v(\bar{\nabla}u\cdot\mathbf{n})\, dl - \nu_i \int_I (\bar{\nabla}v\cdot\mathbf{n})u\, dl + \\ +\beta\nu_i \int_I vu\, dl = -\nu_i \int_I T_m(\bar{\nabla}v\cdot\mathbf{n})\, dl + \beta\nu_i \int_I T_m v\, dl \tag{27}$$

para todo v tal que $v = 0$ en Γ_D. Ésta es la forma débil del problema.

Interpolando la solución como en la sección 3.3 se tiene: $u = \mathbf{Nu}$, $\nabla u = \mathbf{Gu}$, $v = \mathbf{Nv}$ y $\nabla v = \mathbf{Gv}$, donde $\mathbf{N}$ y $\mathbf{G}$ son las matrices que contienen la información del valor de las funciones de forma y de sus derivadas respectivamente.

En el primer término de la ecuación (27) se tiene la derivada de u con respecto al tiempo. Las funciones de forma son constantes respecto del tiempo ya que su valor sólo cambia en el espacio. Por lo tanto, lo que se deriva con respecto al tiempo es el vector $\mathbf{u}$, dejando constante la matriz $\mathbf{N}$.

$$\int_{\Omega_i} v\frac{\partial u}{\partial t}\, d\Omega_i = \int_{\Omega_i} \mathbf{Nv}\frac{\partial \mathbf{Nu}}{\partial t}\, d\Omega_i = \int_{\Omega_i} \mathbf{Nv}\,\mathbf{N}\frac{\partial \mathbf{u}}{\partial t}\, d\Omega_i = \int_{\Omega_i} \mathbf{Nv}\,\mathbf{N}\dot{\mathbf{u}}\, d\Omega_i \tag{28}$$

Los demás términos de la ecuación (27) ya estaban en el ejemplo anterior. Por lo tanto, se puede plantear un sistema $\mathbf{M}\dot{\mathbf{u}} + \mathbf{K}\mathbf{u} = \mathbf{f}$, donde:

$$\mathbf{M} = \int_{\Omega_i} \mathbf{N}^T\,\mathbf{N}\, d\Omega_i \tag{29}$$

$$\mathbf{K} = \nu_i \int_{\Omega_i} \mathbf{G}^T\mathbf{G}\, d\Omega_i - \nu_i \int_I \mathbf{N}^T(\mathbf{n}\cdot\mathbf{G})\, dl - \nu_i \int_I (\mathbf{G}^T\cdot\mathbf{n}^T)\mathbf{N}\, dl + \beta\nu_i \int_I \mathbf{N}^T\mathbf{N}\, dl \tag{30}$$

$$\mathbf{f} = -T_m\nu_i \int_I (\mathbf{G}^T\cdot\mathbf{n}^T)\, dl - T_m\beta\nu_i \int_I \mathbf{N}^T\, dl \tag{31}$$

M es la matriz de masa, **K** es la matriz de rigidez y **f** es el vector de términos independientes. Se puede observar cómo, en este caso, **f** sólo está formado por los términos introducidos en el método de Nitsche, ya que no existe término fuente en la ecuación (25).

Para imponer las condiciones de contorno en la frontera exterior (en el líquido) se debe reducir el sistema como se ha explicado en la sección 3.2, eliminando filas y columnas correspondientes a los nodos pertenecientes a dicho contorno, solucionar el sistema y después añadir a la solución obtenida resolviendo el sistema el valor de las condiciones de contorno.

Para solucionar el sistema $\mathbf{M}\dot{\mathbf{u}} + \mathbf{K}\mathbf{u} = \mathbf{f}$ se aplica el método de Euler explícito o Euler "hacia delante". En este método, se aproxima la derivada de **u** con respecto al tiempo como la diferencia entre la solución en el instante $n+1$ y la solución en el instante n dividido por el tiempo transcurrido entre los dos instantes. El vector **u** que multiplica a la matriz **K** se evalúa en el instante n. El vector **f** también se evaluaría en dicho instante n, pero en este caso es constante en el tiempo. El esquema es el siguiente

$$\mathbf{f} = \mathbf{M}\dot{\mathbf{u}} + \mathbf{K}\mathbf{u} \approx \mathbf{M}\frac{\mathbf{u}^{n+1} - \mathbf{u}^n}{\triangle t} + \mathbf{K}\mathbf{u}^n. \tag{32}$$

Donde $\triangle t$ es el paso de tiempo; el tiempo que transcurre entre los instantes n y $n+1$. En la ecuación (32) se puede despejar $\mathbf{u}^{n+1}$ para obtener la solución en el paso $n+1$ a partir de la solución en el paso n.

$$\mathbf{M}\mathbf{u}^{n+1} = \mathbf{M}\mathbf{u}^n + \triangle t\,(-\mathbf{K}\mathbf{u}^n + \mathbf{f}) \tag{33}$$

Con la ecuación (33) se puede obtener fácilmente la solución en cada paso de tiempo a partir del anterior. Para comenzar se deberá iniciar con una solución inicial a partir de la cual se obtendrán las demás, realizando tantos pasos como se desee. En cada paso se debe solucionar esta ecuación para el líquido y para el sólido. En el método de Euler se trabaja con vectores y matrices reducidos, como se ha explicado en la sección 3.4. Por tanto se obtendrá $\mathbf{u}_R^{n+1}$ a partir de $\mathbf{u}_R^n$. Para ampliar la solución se le añadirán la información de los nodos del contorno Dirichlet.

El método de Euler explícito tiene una condición de estabilidad respecto al paso del tiempo. En un problema como el propuesto, del tipo $u_t = \nu u_{xx}$, se debe cumplir la siguiente condición en el caso de grado 1 y con una matriz

de masa consistente.

$$\frac{\nu \triangle t}{h^2} \leq \frac{1}{6} \tag{34}$$

Como $\nu = \frac{\kappa}{c}$, se tiene que $\nu_1 = 0{,}0111$ y $\nu_2 = 0{,}0196$ son las difusiones del líquido y del sólido respectivamente. Para un tamaño de malla $h = 0{,}125$ cm, que es el tamaño más pequeño, y por lo tanto más restrictivo, que se va a usar, se tienen las siguientes restricciones.

$$\triangle t_1 \leq \frac{h^2}{6\nu_1} = 0{,}234\ s \tag{35}$$

$$\triangle t_2 \leq \frac{h^2}{6\nu_2} = 0{,}132\ s \tag{36}$$

Por lo tanto el paso de tiempo deberá ser siempre menor que 0,132 s, que es el más restrictivo de los dos.

Este problema se va a solucionar de dos maneras: suponiendo una interfaz fija en el tiempo y suponiendo una interfaz móvil. En el segundo caso, los elementos que forman parte de cada dominio cambiarán con el paso del tiempo, por lo que las matrices $\mathbf{M}$, $\mathbf{K}$ y el vector $\mathbf{f}$ cambian en cada paso de tiempo y se deben calcular repetidamente. Además, se debe solucionar un sistema con la matriz de masa M en cada paso, lo que supone un coste computacional alto. Para rebajar este coste computacional se pueden usar dos estrategias.

- Si se utiliza una interpolación lineal se puede trabajar con la matriz "*lumped*", $\mathbf{M}^L$. Esta matriz es diagonal y el coste computacional de hallar su inversa se reduce considerablemente. La matriz $\mathbf{M}^L$ tiene en cada término de la diagonal la suma de los coeficientes de la fila correspondiente; $\mathbf{M}^L(i,i) = \sum_i m_{ij}$. Si se usa una interpolación de grado 2 o mayor, no se puede usar $\mathbf{M}^L$ ya que provoca inestabilidades.
- Con interpolaciones de grados 2 o mayores se utilizar la factorización de Cholesky ya que la matriz de masa es simétrica y definida positiva. Así, se puede descomponer la matriz $\mathbf{M}$ en una matriz triangular inferior y su conjugada, que será triangular superior. $\mathbf{M} = \mathbf{L}\mathbf{L}^T$.

4.1. Interfaz fija

Como ya se ha comentado, en la ecuación (25) existe un término transitorio; la solución **u** depende del tiempo. En este primer apartado se va a considerar que la interfaz no se mueve, su temperatura siempre será la de fusión, $T_m = 0$. Las temperaturas en el sólido y en el líquido cambiarán con el tiempo, dependiendo de las condiciones iniciales y de contorno.

4.1.1. Condiciones de contorno e inicial

Se fijan 10^oC como condición de contorno en el fluido, en la frontera exterior. Para darle al hielo temperatura negativa se pone como condición inicial -5^oC en el centro del hielo. Esta condición en el hielo es condición inicial y no de contorno; por lo tanto no se van a mantener estos -5^oC siempre, como sí se van a mantener los 10^oC en la frontera exterior del fluido. En la Figura 12 se puede ver la condición inicial del problema.

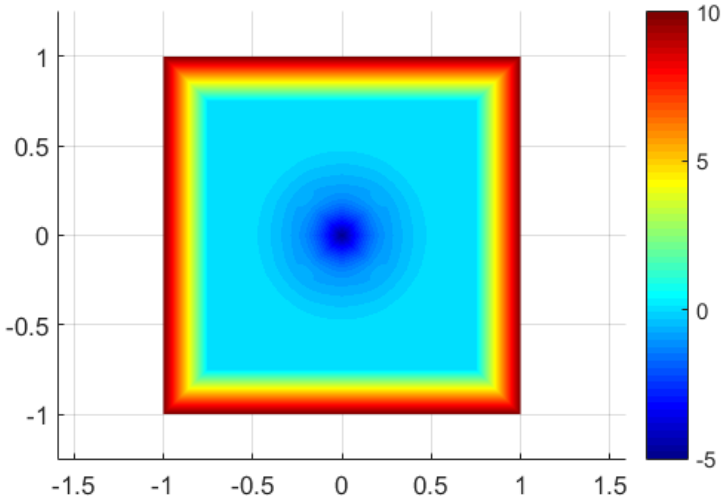

Figura 12: *Condición inicial*

Físicamente cabe esperar que, ya que la temperatura fijada en el líquido es mayor, ésta fluya y suba la temperatura de ambos materiales. Por lo tanto se

espera una temperatura de 0^oC en la interfaz ya que es condición de contorno fijada, una progresión desde 10^oC a 0^oC en el líquido y en el sólido se espera una progresiva desaparición de las temperaturas negativas hasta que, a largo plazo, la temperatura sea de 0^oC en todo el sólido.

De hecho, la situación a la que tiende el problema según el transcurso del tiempo es a la del problema estacionario, eliminando de la ecuación (25) el primer término, el que depende del tiempo.

4.1.2. Solución estacionaria

Si se elimina de (25) la parte transitoria el problema quedaría de la siguiente manera.

$$\begin{cases} -\bar{\nabla} \cdot (\nu_i \bar{\nabla} \mathbf{u}) = 0 \quad en \quad \Omega_i \quad con \quad i = 1, 2 \\ u = T_m = 0^oC \quad en \quad I \\ u = u_D = 10^oC \quad en \quad \Gamma_D \end{cases} \tag{37}$$

Con las condiciones de contorno explicadas en el apartado anterior, aquí no hay condición inicial ya que sirve únicamente para iniciar el método de Euler, la solución $\mathbf{u}$ se obtendría resolviendo un sistema $\mathbf{Ku} = \mathbf{f}$. Se obtendría una solución para el líquido, $\mathbf{u}_1$ y otra para el sólido, $\mathbf{u}_2$.

En la Figura 13 se puede ver esta solución.

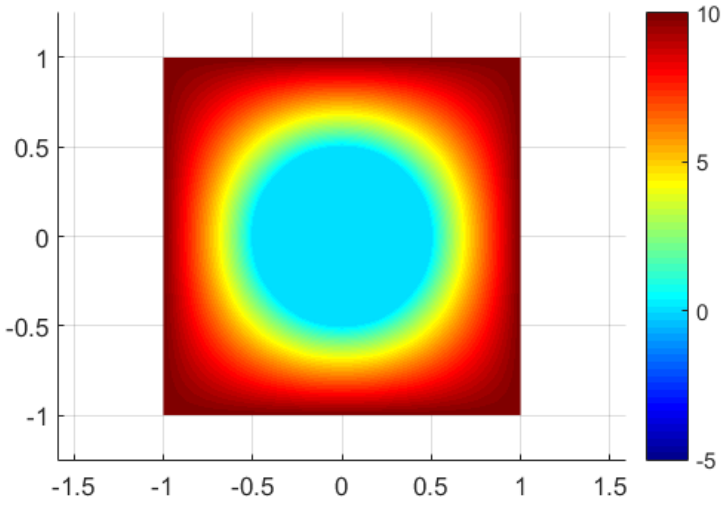

Figura 13: *Solución estacionaria*

4.1.3. Solución transitoria

Se realizan varios pasos con el método de Euler explícito, como condición inicial la Figura 12. Como paso de tiempo se ha usado $\triangle t = 0{,}05$ s. Los resultados son los siguientes.

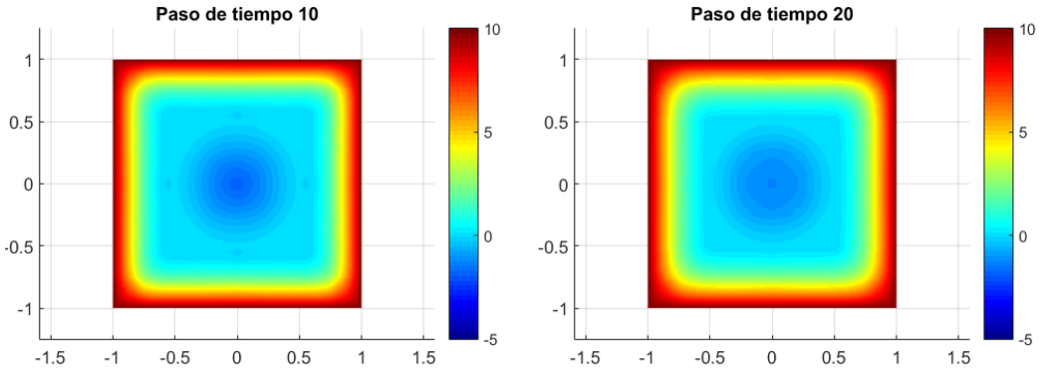

Figura 14: *Pasos de tiempo 10 (0.5s) y 20 (1s).*

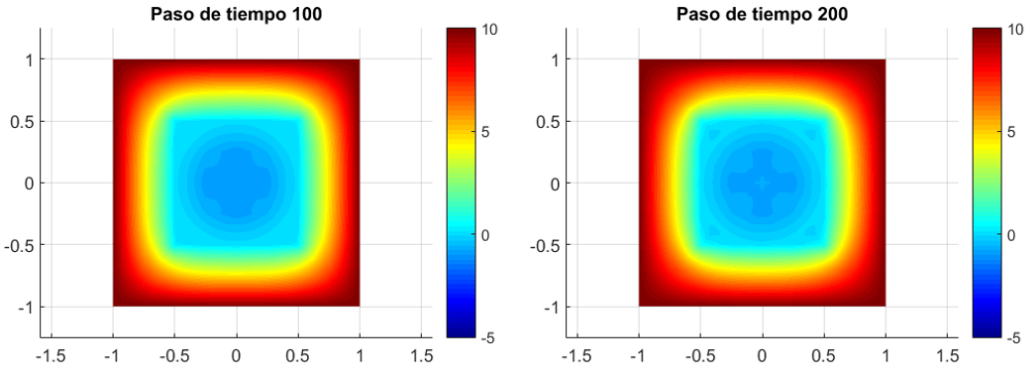

Figura 15: *Pasos de tiempo 100 (5s) y 200 (10s).*

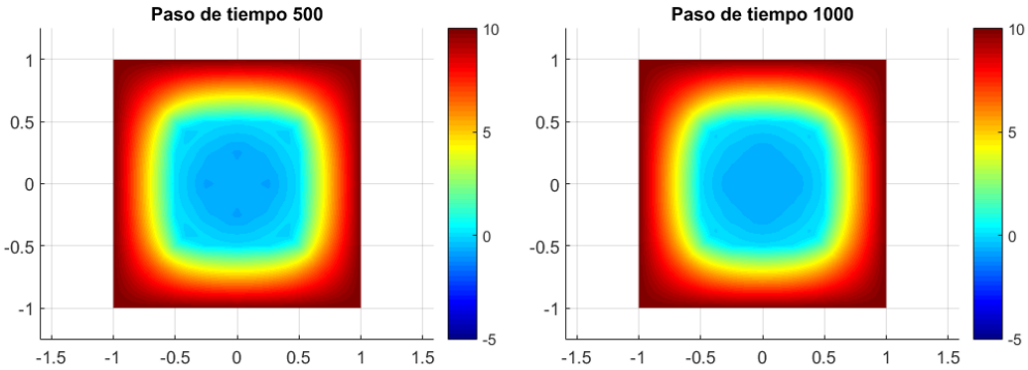

Figura 16: *Pasos de tiempo 500 (25s) y 1000 (50s).*

En las Figuras 14,15 y 16 se observa perfectamente cómo, con el transcurso del tiempo, la solución se va alejando de la solución inicial y cada vez se parece más a la estacionaria. La temperatura negativa en el sólido va tendiendo a 0 con el tiempo y, con 500 pasos de tiempo (25 segundos transcurridos), prácticamente ya es nula, todas las temperaturas en el sólido rondan los 0^oC. En el líquido se observa cómo las temperaturas positivas fluyen hacia la interfaz y, poco a poco, tienden a formar una transición desde los 10^oC en el contorno exterior hasta los 0^oC en la interfaz en la que, las isotermas se van haciendo circulares a medida que se acercan a la interfaz.

4.2. Interfaz móvil. Problema de Stefan

En este apartado se soluciona el problema de Stefan. El hielo y el agua líquida están separados por una interfaz móvil a 0^oC. Esta interfaz puede ganar temperatura, haciendo que el líquido se expanda y por tanto la interfaz se mueva hacia la parte del sólido o puede pasar todo lo contrario, que sea el sólido el que se expanda. Depende de las condiciones de contorno e iniciales que se impongan en el problema.

Para modelar el movimiento de la interfaz se debe actualizar el level-set en cada paso de tiempo. Así, el dominio de cada fase cambiará con el tiempo y será necesario calcular las matrices $\mathbf{M}$, $\mathbf{K}$ y el vector $\mathbf{f}$ en cada paso de tiempo.

4.2.1. Condiciones de contorno e iniciales

Sólo se imponen condiciones de contorno en la frontera exterior, 10^oC. En el hielo se imponen las mismas condiciones iniciales que antes, -5^oC en el centro del hielo. En el líquido se impone como condición inicial la solución estacionaria. En el apartado de interfaz fija no se podía poner como condición inicial la estacionaria ya que, si en la ecuación(**??**) introduces como $\mathbf{u}_n$ la solución estacionaria para el líquido, $\mathbf{u}_{n+1}$ será exactamente igual que $\mathbf{u}_n$ y la solución no cambiará. Sin embargo, con una interfaz móvil la solución si irá cambiando porque, con las condiciones iniciales que se han impuesto, se espera que el líquido se vaya expandiendo y el dominio Ω_1 cada vez será mayor. A su vez, el vector $\mathbf{u}_1$ irá cambiando de dimensión cada vez que el líquido se expanda y el sólido colapse.

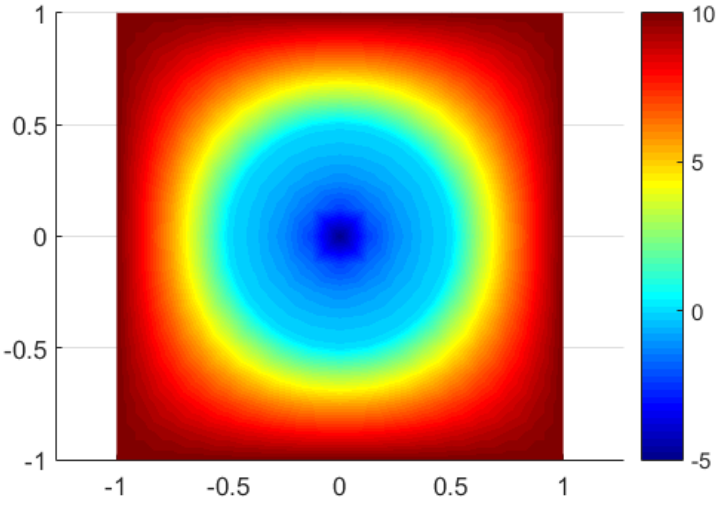

Figura 17: *Solución inicial*

4.2.2. Actualización el Level-Set

Para actualizar el level-set lo primero que se debe hacer es calcular la velocidad normal en la interfaz, v_n. En un punto cualquiera de la interfaz la velocidad normal se calcula con la ecuación

$$v_n = \frac{1}{L}(\kappa_1 \nabla \mathbf{u}_1 \cdot \mathbf{n}_1 + \kappa_2 \nabla \mathbf{u}_2 \cdot \mathbf{n}_2). \tag{38}$$

En esta ecuación L es el calor latente de fusión, la cantidad de energía necesaria para cambiar de fase. En el caso del agua $L = 19{,}2\ cal/cm^3$. κ_1 y κ_2 son las conductividades térmicas del líquido y del sólido respectivamente y $\nabla \mathbf{u}_1$ y $\nabla \mathbf{u}_2$ los gradientes de las temperaturas en el líquido y en el sólido. $\mathbf{n}_1$ es el vector normal exterior al líquido y $\mathbf{n}_2$ el exterior al sólido. Tomando como referencia el vector normal exterior al líquido se puede reescribir (38) de la siguiente manera

$$v_n = \frac{1}{L}(\kappa_1 \nabla \mathbf{u}_1 - \kappa_2 \nabla \mathbf{u}_2) \cdot \mathbf{n}_1. \tag{39}$$

Una vez calculada la velocidad normal se debe calcular la velocidad específica. Ésta está dada por su valor en los nodos. La componente "i" de

este vector de velocidades específicas será el valor de la velocidad normal en el punto de la interfaz más cercano al nodo "i". Por ejemplo, para obtener la primera componente del vector primero se traza una perpendicular a la interfaz desde el nodo 1, obteniendo el punto de la interfaz más cercano a dicho nodo, y el valor del primer término del vector v_e es la velocidad normal en ese punto. Y así con cada uno de los nodos de la malla.

En la práctica, para calcular el vector v_e primero se generan muchos puntos a lo largo de la interfaz. Se calcula la velocidad normal en cada uno de estos puntos y después a cada nodo del dominio se le asigna el punto más cercano de los ya calculados en la interfaz. Si se han generado suficientes puntos, la línea que une el nodo con su puntos más cercano en la interfaz será prácticamente una perpendicular a ésta. Así pues, una vez se ha calculado la velocidad normal en estos puntos, se puede generar el vector v_e.

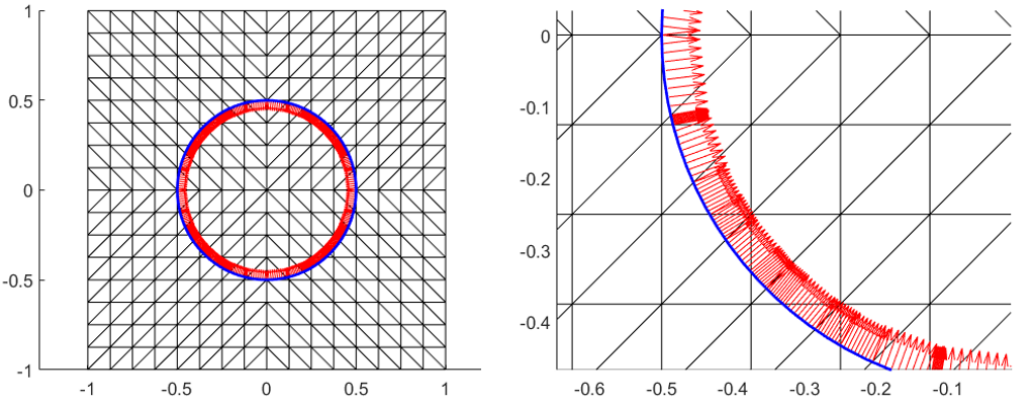

Figura 18: *A la izquierda se ven las velocidades normales en la interfaz. A la derecha se ve una ampliación de una parte.*

La velocidad normal es un escalar, no un vector. En la Figura 18 se puede observar la velocidad normal multiplicada por la normal del líquido, $\mathbf{n}_1$. Como se observa, las velocidades son positivas ya que van en dirección de la normal $\mathbf{n}_1$. Por lo tanto, tal y como se esperaba debido a las condiciones de contorno e iniciales, será el líquido el que se expanda y la interfaz se moverá de tal manera que encerrará cada vez menos cantidad de sólido.

Una vez calculado el vector de velocidades específicas a partir de las velocidades normales, se puede actualizar el level-set con la ecuación

$$LS^{n+1} = LS^n + v_e \cdot \triangle t. \tag{40}$$

Así, se obtiene el level-set en el paso $(n+1)$ a partir del level-set en el anterior, el paso de tiempo $(\triangle t)$ y la velocidad específica, v_e.

4.2.3. Proyección de la solución

La principal consecuencia de que se mueva el level-set es que el dominio del líquido aumenta y el del sólido disminuye en cada paso de tiempo. Por tanto, los nodos que pertenecen a cada uno de los dominios no son constantes en el tiempo.

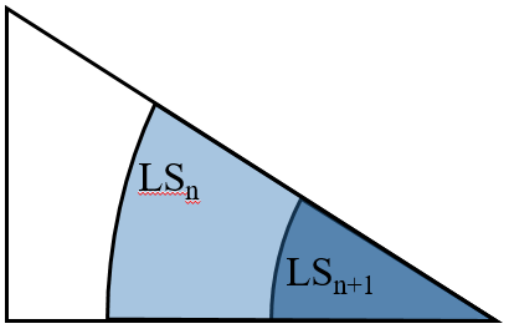

Figura 19: *Elemento cortado por el level-set en el paso n y el level-set en el paso n+1.*

En la Figura 20 se observan 3 regiones. La parte de la izquierda, blanca, es parte del líquido. La región de la derecha, azul oscura, pertenece al sólido. La parte del elemento entre los dos level-sets, azul claro, es el área de fusión, la parte del dominio que pertenece al sólido en el paso n y al líquido en el paso $(n+1)$.

Siendo φ_1 la función del level-set en el paso de tiempo n y φ_2 la función del level-set en el paso de tiempo $n+1$, matemáticamente se pueden distinguir las tres regiones de la siguiente manera.

$$\begin{cases} \varphi_1(\mathbf{x}) > 0 & \& & \varphi_2(\mathbf{x}) > 0 & \mathbf{x} \in \Omega_1^n \\ \varphi_1(\mathbf{x}) < 0 & \& & \varphi_2(\mathbf{x}) > 0 & \mathbf{x} \in \Omega_1^{n+1} \setminus \Omega_1^n \\ \varphi_1(\mathbf{x}) < 0 & \& & \varphi_2(\mathbf{x}) < 0 & \mathbf{x} \in \Omega_2 \end{cases} \tag{41}$$

Ω_1^n es el dominio inicial del líquido. $\Omega_1^{n+1} \setminus \Omega_1^n$ es el área de fusión y Ω_2 es el dominio del sólido.

Para el líquido, en el paso n, se tiene la solución $\mathbf{u}^n$, la solución reducida $\mathbf{u}_R^n$ y el level-set, LS^n. El objetivo es calcular la solución en el paso $(n+1)$, $\mathbf{u}^{n+1}$. Para ello, se debe solucionar el sistema $\mathbf{M}\mathbf{u}^{n+1} = \mathbf{M}\mathbf{u}^n + \triangle t\ (-\mathbf{K}\mathbf{u}^n + \mathbf{f})$. Pero en este sistema se deben introducir los vectores y las matrices reducidas. A partir de $\mathbf{u}_R^n$ se obtendrá $\mathbf{u}_R^{n+1}$. El problema es que los vectores no son de la misma dimensión; para solucionar este inconveniente se debe proyectar el vector $\mathbf{u}^n$, añadiendo información de los nodos pertenecientes al área de fusión, obteniendo $\mathbf{u}_{NEW}^n$. Una vez obtenido este vector, se reduce eliminando las componentes pertenecientes al contorno y a la parte del dominio correspondiente al sólido teniendo en cuenta el level-set en el paso de tiempo $(n+1)$. Ahora ya se puede solucionar el sistema y obtener $\mathbf{u}_R^{n+1}$ ya que la dimensión de los vectores coincide. Al vector obtenido se le añade la información de las condiciones de contorno y se obtiene $\mathbf{u}^{n+1}$.

Para obtener $\mathbf{u}_{NEW}^n$ se realiza un ajuste por mínimos cuadrados.

$$\mathbf{u}_{NEW}^n \ t.q\ \text{mín} \int_{\Omega_1^n} (\mathbf{u}^n - \mathbf{u}_{NEW}^n)^2 \, d\Omega + \int_{\Omega_1^{n+1} \setminus \Omega_1^n} (\mathbf{u}_{NEW}^n - T_m)^2 \, d\Omega \qquad (42)$$

Para el sólido, sin embargo, no hay que hacer lo mismo. Como se sabe que el dominio del sólido siempre se reducirá, simplemente hay que aplicar las condiciones de contorno correspondientes en cada paso, teniendo en cuenta el nuevo level-set. Así se reducirá el vector correctamente y se podrá obtener $\mathbf{u}_R^{n+1}$ a partir de $\mathbf{u}_R^n$.

En cada paso de tiempo, además, hay que calcular las matrices $\mathbf{K}$, $\mathbf{M}$ y el vector $\mathbf{f}$ tanto para el líquido como para el sólido. Por lo tanto, el gasto computacional es muchísimo mayor que si se mantiene la interfaz fija.

4.2.4. Resultados

Como se ha explicado, el coste computacional requerido para calcular cada paso de tiempo es muy grande. Por tanto, no se pueden calcular tantos pasos como en el problema con la interfaz fija, pero sí se puede mostrar resultados de cómo evoluciona el problema al principio y comprobar que los resultados son los esperados.

En la Figura 20 se observa el movimiento de la interfaz tras 5 pasos de tiempo con un $\triangle t = 0{,}07$ s. En 0.35s la evolución no es muy grande pero sí se ve cómo la interfaz comienza a moverse reduciendo el dominio del hielo.

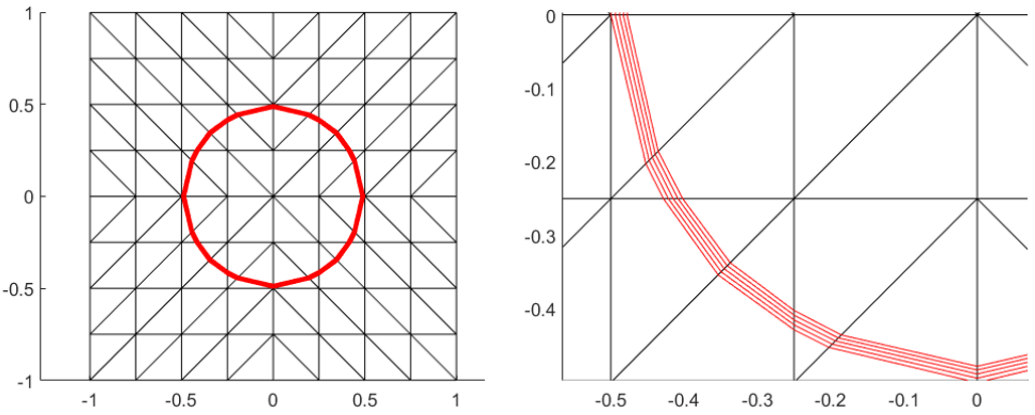

Figura 20: *A la izquierda, evolución del Level-Set en 5 pasos de tiempo. A la derecha, una ampliación de una parte*

Tras 20 pasos de tiempo con $\triangle t = 0{,}07$ s han transcurrido $1,4$ s. La evolución no se nota tanto como en el problema de interfaz fija, en el que se estudiaban 50 segundos pero sí se puede apreciar cómo el hielo comienza a fundirse.

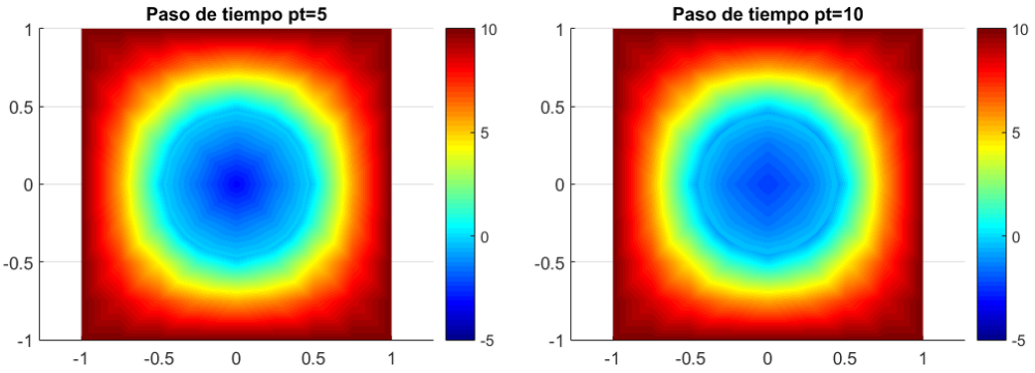

Figura 21: *Solución en los pasos de tiempo 5 (0.35 s) y 10 (0.7 s)*

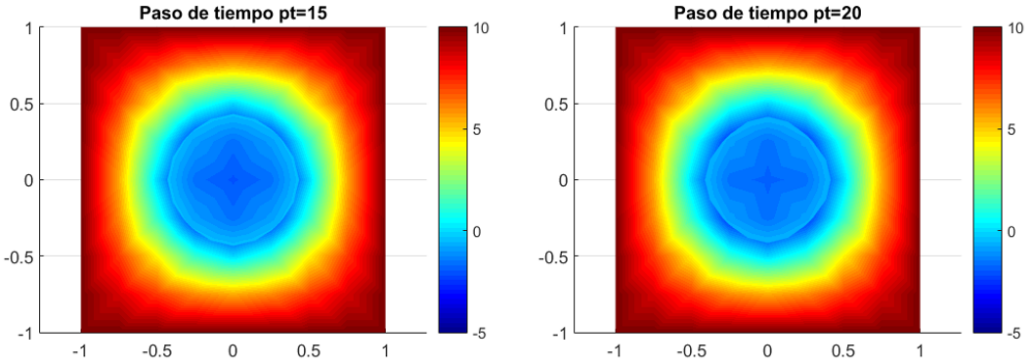

Figura 22: *Solución en los pasos de tiempo 15 (1.05 s) y 20 (1.4 s)*

Como se ve en las Figuras 21 y 23 las temperaturas en el hielo van aumentando con el paso del tiempo. Las temperaturas en el sólido son prácticamente similares ya que la solución inicial que se ha impuesta es la estacionaria. Sí que se aprecia un poco que la isoterma de 0^oC, la interfaz, se va haciendo más pequeña con el tiempo. De hecho en la Figura 24 se puede observar el level-set inicial y el level-set tras 1.4 segundos transcurridos.

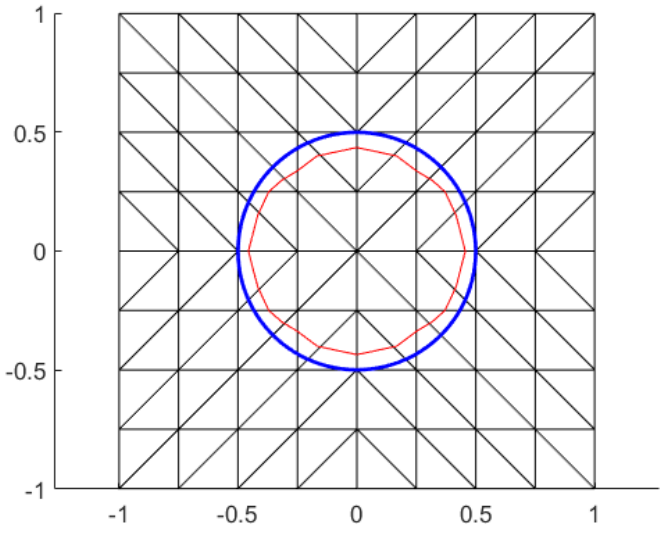

Figura 23: *Level-set inicial (en azul) y level-set transcurridos 1.4 s (en rojo)*

En ocasiones, al solucionar problemas transitorios con el método de Euler, existen errores que se van propagando con los pasos de tiempo de tal forma que, pasados unos pasos de tiempo, la solución no es fiable. Para comprobar que no existen problemas de este tipo, se ha obtenido la solución a los 1.4 s pero usando el doble de pasos (40 pasos) con $\triangle t = 0{,}07/ = 0{,}035$ s y con el triple de pasos (60) con $\triangle t = 0{,}07/3 = 0{,}023$ s. También se han usado diferentes mallas y diferentes grados de interpolación de la solución y ésta siempre ha sido similar, por lo que se puede concluir que los resultados son fiables.

Aparte del gran coste computacional al calcular muchos pasos de tiempo, también se incrementa la posibilidad de que la interfaz corte a un elemento de una de las dos formas que se presentan en la Figura 24, en el que el ratio de los dos dominios en el elemento cortado es muy pequeño. Si esto ocurre, el programa desarrollado en este trabajo no es capaz de solucionar estos problemas. Para solventarlo, existen dos técnicas. La primera consiste en mover los nodos de los elementos para que la interfaz corte de una forma favorable. La segunda, algo más sofisticada, consiste en realizar una estabilización. Para ello se debe modificar la forma débil del problema añadiendo términos que solucionan este problema.

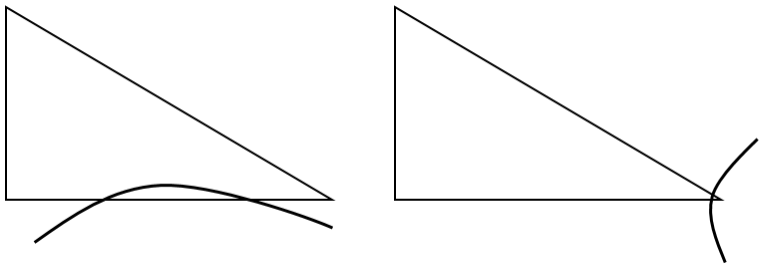

Figura 24: *Elementos cortados por la interfaz de forma problemática*

5. Conclusiones

En la primera parte se ha solucionado una ecuación estacionaria en derivadas parciales, planteada en un dominio con un agujero en el medio. Se ha usado una función level-set para identificar la interfaz I (el borde del agujero). No es necesario que la malla se ajusta a esta interfaz como es habitual

en X-FEM. Para imponer condiciones de contorno en la interfaz se ha usado el método de Nitsche. El problema se ha solucionado para varias mallas, con distintos tamaños de elemento, y para varios grados de interpolación para estudiar la convergencia del método. Se ha comprobado que el error cometido al comparar la solución numérica, la obtenida con X-FEM, y la solución analítica sigue la relación $E \sim Ch^{p+1}$ siendo E el error, C una constante, h el tamaño de elemento y p el grado de interpolación. Por lo tanto, se puede afirmar que el método propuesto es consistente ya que los resultados obtenidos son fiables y se comportan de la manera esperada.

En la segunda parte se ha propuesto una EDP transitoria, definida en un dominio bimaterial: agua líquida y hielo separados por una interfaz I. Se ha solucionado este problema de dos formas diferentes. Primero, se ha considerado una interfaz fija en el tiempo y después se ha resuelto el llamado problema de Stefan, considerando una interfaz móvil. Al método desarrollado en la primera parte del trabajo se han añadido cálculos para tener en cuenta la parte transitoria del problema: obtención de la matriz de masa, método de Euler, etc.

En el problema de Stefan los dominios del agua líquida y del hielo no son constantes en el tiempo. Para poder mover la interfaz en cada paso de tiempo se ha calculado la velocidad normal en la interfaz, se ha actualizado la función level-set y se ha proyectado la solución para tener en cuenta los cambios de dominio.

Tanto los resultados obtenidos con una interfaz fija como con una interfaz móvil concuerdan totalmente con lo esperado físicamente. Considerando la interfaz fija se ha obtenido la solución para un tiempo mayor, ya que el coste computacional es mucho menor. Con la interfaz móvil se ha obtenido la solución pasados $1,4$ segundos pero se han utilizado diferentes pasos de tiempo, diferentes mallas y diferentes grados de interpolación y los resultados obtenidos son similares. Por todo esto, se puede concluir que el método implementado en este trabajo es fiable.

Referencias

[1] Sonia Fernández Méndez, Antonio Huerta, *Imposing essential boundary conditions in mesh-free methods*, 2003.

[2] Esther Sala Lardies, Sonia Fernández Méndez, Antonio Huerta, *Optimally convergent high-order X-FEM for problems*, 2012.

[3] Ceren Gürkan, Sonia Fernández Méndez, Esther Sala Lardies, Martin Kronbichler, *eXtended Hybridizable Discontinous Galerkin (X-HDG) for void problems.*

[4] H. Ji, D. Chopp, J. E. Dolbow, *A hybrid extended finite element/level set method for modeling phase transformations*, 2001.

6. Anejos

mainXFEM

```
%Éste es el código principal del problema estacionario. Soluciona la EDP
utilizando distintas funciones para obtener las matrices necesarias.

clear all, clc, close all
setpath

diff=1; %Valor de la difusión
global centre radius
centre=[0,0]; radius=0.5; %Centro y radio del agujero

%_________________PREPROCESO
%Grado de interpolación, creación de la malla (X,T), X es el vector que
contiene las coordenadas de los nodos y T es la matriz de conectividad,
función level-set
degree = 2;
numMallas = 3;
for meshNum= numMallas

    meshName = sprintf('mesh%d_P%d.dcm',meshNum,degree);  fprintf('\n
Mesh %s:\n',meshName);

    if all(meshName(end-2:end)=='dcm') GenerateMatFileFromEZ4U(['Meshes/'
meshName]); end
    load(['Meshes/' meshName(1:end-3) 'mat']);
    X = 2*X-1; %mesh on [-1,1]^2
    figure(1),clf
    plotMesh(X,T)

    %Elemento de Referencia
    referenceElement = createReferenceElement(1,elemInfo.nOfNodes);

    %Level-set
    LS = EvaluateLS(X);
    t=linspace(0,2*pi,101); figure(1), hold on,
plot(radius*cos(t),radius*sin(t),'LineWidth',2), hold off

    %Elements contiene información de los distintos tipos de elementos:
estándar y corados
    Elements = SetElements(T,LS,[1,0],referenceElement);

    %Condiciones de contorno
    x = X(:,1); y = X(:,2); tol=1.e-10;
    nodesCCD = find(abs(x+1)<tol|abs(x-1)<tol|abs(y+1)<tol|abs(y-1)<tol);
%Identificación de los nodos del contorno
    figure(1), hold on,
plot(x(nodesCCD),y(nodesCCD),'bo','MarkerSize',16); hold off
    XnodesCCD = X(nodesCCD,:);  %Coordenadas de los nodos del contorno
    uCCD = boundaryValue(XnodesCCD); %Valor de contorno
```

```
    %______OBTENCIÓN DEL SISTEMA
standardElements = [Elements.D1];

[K,f]=computeSystemLaplace(X,T(standardElements,:),referenceElement,diff)
; %Obtención de la matriz de rigidez K y del vector f para elementos
estándar

[Kcut,fcut]=computeSystemLaplaceCutElements(LS,X,T(Elements.Int,:),refere
nceElement,diff); %Obtención de la matriz de rigidez K y del vector f
para elementos cortados
    K=(K+Kcut); f=f+fcut;

     Beta = 100; %Parámetro de Nitsche

[K,f]=computeSystemLaplaceCutElementsNitsche(Beta,K,f,LS,X,T(Elements.Int
,:),referenceElement,diff); %Obtención de la matriz de rigidez K y del
vector f añadiendo los términos de Nitsche

    %Nodos en el agujero
    nodesVoid = find(sum(abs(K))<1.e-14)';
    nodesCCD = [nodesCCD ; nodesVoid];
    uCCD = [uCCD ; zeros(size(nodesVoid))];

    %Imposición de las condiciones de contorno Dirichlet, reducción del
sistema
    notCCD= setdiff(1:size(X,1),nodesCCD); %actual degrees of freedom
(not boundary nodes)
    f = f(notCCD)-K(notCCD,nodesCCD)*uCCD;
    K=K(notCCD,notCCD);

    %Solución del sistema
    sol=K\f;
    %Nodal values
    u = zeros(size(X,1),1);
    u(notCCD) = sol; u(nodesCCD) = uCCD;


    %__________POSTPROCESO

    %Dibujar la solución

[Xl,Tl,ul]=XFEMtoFineLinearApproximation(LS,X,T,u,10,referenceElement);
    figure(4), clf, trisurf(Tl,Xl(:,1),Xl(:,2),ul), shading interp,
view(2), axis equal, colorbar, colormap jet,set(gca, 'FontSize',12.5)

    %Cálculo del error, norma L2
    L2errorStd =
computeL2Norm(referenceElement,X,T(standardElements,:),u,@analyticalSolut
ion); %para elementos estándar
    L2errorCut =
computeL2NormCut(LS,referenceElement,X,T(Elements.Int,:),u,@analyticalSol
ution); %para elementos cortados
    L2error = (L2errorStd^2+L2errorCut^2)^0.5;
    fprintf('  L2 error = %0.2e\n',L2error)
    end
```

computeSystemLaplaceCutElements

```
%Este código cálcula la matriz de rigidez K y el vector f para los
elementos cortados por la interfaz.

function [K,f] =
computeSystemLaplaceCutElements(LS,X,T,referenceElement,diff)

nOfNodes = size(X,1); %número de nodos
nOfElements = size(T,1); %número de elementos

K=spalloc(nOfNodes,nOfNodes,5*nOfNodes); %inicialización matriz global
f=zeros(nOfNodes,1); %inicialización vector global

nDeg=referenceElement.degree; %grado de interpolación

%Cuadratura para un triángulo y un cuadrilátero estándar
[zgp_qua,wgp_qua] = GaussLegendreCubature2Dquad(nDeg+1);
zgp_tri = referenceElement.IPcoordinates;
wgp_tri = referenceElement.IPweights;


%Bucle en elementos
for i=1:nOfElements
    Te=T(i,:); %nodos en el elementonodes in the element
    Xe=X(Te,:); %coordenadas de los nodos del elemento
    LSe=LS(Te); %level-set

%Obtención de puntos y pesos de integración únicamente en Omega_1;(LS>0)
    [GaussPoints,GaussWeights,n1,n2,PtsInt,CutFaces] =
ModifyQuadrature(LSe,referenceElement,zgp_tri,wgp_tri,zgp_qua,wgp_qua);
    GaussPoints=GaussPoints(1:n1,:); GaussWeights=GaussWeights(1:n1);


%Valor de las funciones de forma y derivadas en los puntos de Gauss
anteriores
shapeFunctions=computeShapeFunctionsAtPoints(nDeg,referenceElement.NodesC
oord,GaussPoints);
    N = shapeFunctions(:,:,1)';
    Nxi = shapeFunctions(:,:,2)';
    Neta =shapeFunctions(:,:,3)';

%Obtención de K elemental y f elemental
[Ke,fe]=computeElementalMatricesCutElements(LSe,Xe,GaussPoints,GaussWeigh
ts,N,Nxi,Neta,diff);
    K(Te,Te)=K(Te,Te)+Ke; %ensamblaje
    f(Te) = f(Te) + fe; %ensamblaje
end

%_____________________________________
%Obtención de matriz y vector elementales
function [Ke,fe]=
computeElementalMatricesCutElements(LSe,Xe,GaussPoints,GaussWeights,N,Nxi
,Neta,diff)
```

```
nOfNodes = size(Xe,1);
Ke=zeros(nOfNodes);
fe=zeros(nOfNodes,1);
xe = Xe(:,1); ye = Xe(:,2); %coordenadas 'x' e 'y' de los nodos del
elementos

%Bucle en puntos de integración
for k=1:length(GaussWeights)

    %Funciones de forma y derivadas
    Nk=N(k,:);
    dNkdxi=Nxi(k,:);
    dNkdeta=Neta(k,:);
    xk = Nk*Xe;

    %Jacobiano de la transformación paramétrica
    J = [dNkdxi*xe dNkdxi*ye;dNkdeta*xe dNkdeta*ye];

    %Derivadas de las funciones de forma con respect a 'x' e 'y'
    Gk = J\[dNkdxi;dNkdeta];

    %Integración numérica
    dxy=GaussWeights(k)*det(J);
    Ke = Ke + diff*Gk'*Gk*dxy; %Información de la forma débil
    fe = fe + source(xk,diff)*Nk'*dxy; %Información de la forma débil
end
```

computeSystemLaplaceCutElementsNitsche

```
%Este código calcula la matriz K y f añadiendo los términos que introduce
el método de Nitsche para imponer condiciones de contorno en la interfaz.

function [K,f]=
computeSystemLaplaceCutElementsNitsche(beta,K,f,LS,X,T,referenceElement,d
iff)

nOfNodes = size(X,1); %número de nodos
nOfElements = size(T,1); %número de elementos

%Funciones de forma y derivadas 1D, puntos y pesos de integración
N1D = referenceElement.N1d;
dNds1D = referenceElement.N1dxi;
z1D = referenceElement.IPcoordinates1d;
w1D = referenceElement.IPweights1d; ngauss=length(z1D);

nDeg=referenceElement.degree; %grado de interpolación

%Cuadratura para un triángulo y un cuadrilátero estándar
[zgp_qua,wgp_qua] = GaussLegendreCubature2Dquad(nDeg+1);
zgp_tri = referenceElement.IPcoordinates;
wgp_tri = referenceElement.IPweights;

% Bucle en elementos
for iElem=1:nOfElements
    Te=T(i,:); %nodos en el elemento
    Xe=X(Te,:); %coordenadas de los nodos del elemento
    LSe=LS(Te); %level-set

%Otención de los nodos de la interfaz en el elemento de referencia
    [GaussPoints,GaussWeights,n1,n2,nodesInterfaceReferenceElem,CutFaces]
= ModifyQuadrature(LSe,referenceElement,zgp_tri,wgp_tri,zgp_qua,wgp_qua);
    n=length(nodesInterfaceReferenceElem)/2;
nodesInterfaceReferenceElem=reshape(nodesInterfaceReferenceElem,2,n)';

%Valor de las funciones de en los nodos de la interfaz
shapeFunctions=computeShapeFunctionsAtPoints(nDeg,referenceElement.NodesC
oord,nodesInterfaceReferenceElem);
    N = shapeFunctions(:,:,1)';
    nodesInterfacePhysical= N*Xe; %nodos de la interfaz en el elemento
fisicointerface

    xieta_integrationPoints1D = N1D*nodesInterfaceReferenceElem;
%coordenadas de los puntos de integración en el elemento de referencia

%Valor de las funciones de forma y derivadas en los puntos de integración
en el elemento de referencia

shapeFunctions=computeShapeFunctionsAtPoints(nDeg,referenceElement.NodesC
oord,xieta_integrationPoints1D);
    N = shapeFunctions(:,:,1)'; Nxi = shapeFunctions(:,:,2)'; Neta
=shapeFunctions(:,:,3)';
```

```
%Obtención de la derivada de las funciones de forma en dichos puntos con
respecto a 'x' e 'y'

    J11 = Nxi*Xe(:,1); J12 = Nxi*Xe(:,2);
    J21 = Neta*Xe(:,1); J22 = Neta*Xe(:,2);
    detJ = J11.*J22-J12.*J21;

    invJ11 = spdiags(J22./detJ,0,ngauss,ngauss);
    invJ12 = spdiags(-J12./detJ,0,ngauss,ngauss);
    invJ21 = spdiags(-J21./detJ,0,ngauss,ngauss);
    invJ22 = spdiags(J11./detJ,0,ngauss,ngauss);


    Nx = invJ11*Nxi + invJ12*Neta;
    Ny = invJ21*Nxi + invJ22*Neta;

%Obtención de K elemental y f elemental, para cada uno de los términos
añadidos en Nitsche

[K2e,K3e,K4e,f2e,f3e]=computeElementalMatricesCutElements(LSe,Xe,GaussPoi
nts,GaussWeights,ngauss,N1D,dNds1D,N,Nx,Ny,nodesInterfacePhysical,w1D,dif
f);
    K(Te,Te)=K(Te,Te)-K2e-K3e+beta*K4e; %ensamblaje
    f(Te) = f(Te) - f2e + beta*f3e; %ensamblaje
end

%_____________________________________________
%Obtención de matriz y vector elementales

function [K2e,K3e,K4e,f2e,f3e]=
computeElementalMatricesCutElements(LSe,Xe,GaussPoints,GaussWeights,ngaus
s,N1D,dNds1D,N,Nx,Ny,nodesInterfacePhysical,w1D,diff)

nOfNodes = size(Xe,1);
K2e=zeros(nOfNodes);
K3e=zeros(nOfNodes);
K4e=zeros(nOfNodes);
f2e=zeros(nOfNodes,1);
f3e=zeros(nOfNodes,1);
xe = Xe(:,1); ye = Xe(:,2); %coordenadas 'x' e 'y' de los nodos del
elementos

%Bucle en puntos de integración

    for g = 1:ngauss
        N1D_g = N1D(g,:);
        N2D_g = N(g,:); N2D_dx_g = Nx(g,:); N2D_dy_g = Ny(g,:);
        G_g = [N2D_dx_g;N2D_dy_g];
        dxds_g = dNds1D(g,:);
        xg = N1D_g*nodesInterfacePhysical; %coordenadas de los puntos de
integración en el elemento físico

    %Integración numérica
        dxds_g = dxds_g*nodesInterfacePhysical;
        norma = norm(dxds_g);
        dl = w1D(g)*norma;
```

```
    t = dxds_g/norma;
    n = [t(2),-t(1)];
%Información de la forma débil
    K2e = K2e + diff*N2D_g'*(n*G_g)*dl;
    K3e = K3e + diff*(G_g'*n')*N2D_g*dl;
    K4e = K4e + diff*(N2D_g'*N2D_g)*dl;
    f2e = f2e + diff*boundaryValue(xg)*(G_g'*n')*dl;
    f3e = f3e + diff*boundaryValue(xg)*N2D_g'*dl;
end
```

mainTransientNotMovingInterface

```
%Éste es el código principal que soluciona la EDP transitoria considerando
una interfaz fija.
%El preproceso es igual que en el código mainXFEM. Crea la malla, elemento
de referencia, etc.
%Se divide el código en líquido y en sólido. Hallando las matrices de cada
material, imponiendo condiciones de contorno, solucionando por separado
mediante el método de Euler, etc.

clear all, clc, close all
setpath

global centre radius
centre=[0,0]; radius=0.5; %Centro y radio de la parte sólida.

%_________________PREPROCESO
%Malla (X, T) sobre el dominio y grado de interpolación
degree = 2;
numMallas = 3;
meshNum=numMallas;
meshName = sprintf('mesh%d_P%d.cdm',meshNum,degree);

if all(meshName(end-2:end)=='dcm') GenerateMatFileFromEZ4U(['Meshes/'
meshName]); end
load(['Meshes/' meshName(1:end-3) 'mat']);
X = 2*X-1;
figure(1),clf
plotMesh(X,T)

%Elemento de referencia
referenceElement = createReferenceElement(1,elemInfo.nOfNodes);
Beta = 1000*(2^meshNum);

%Propiedades de los materiales
kl=6.9e-3; %Conductividad térmica del líquido
cl=0.62; %Capacidad calorífica volumétrica del líquido
ks=9.6e-3; %Conductividad térmica del sólido
cs=0.49; %Capacidad calorífica volumétrica del sólido
diffl=kl/cl;
diffs=ks/cs;
L=19.2; %calor latente de fusión


%_____________________________________________________________________
%LÍQUIDO

%Información del Level-set
LS = EvaluateLS(X);
t=linspace(0,2*pi,101); figure(1), hold on,
plot(radius*cos(t),radius*sin(t),'LineWidth',2), hold off
Elements = SetElements(T,LS,[1,0],referenceElement);
%Nodos en el contorno
x = X(:,1); y = X(:,2); tol=1.e-10;
nodesCCDl = find(abs(x+1)<tol|abs(x-1)<tol|abs(y+1)<tol|abs(y-1)<tol);
```

```
figure(1), hold on, plot(x(nodesCCDl),y(nodesCCDl),'bo','MarkerSize',16);
hold off
XnodesCCDl = X(nodesCCDl,:);  %coordenadas de los nodos del contorno
uCCDl = 10*ones(size(XnodesCCDl(:,1))); %valor del contorno


%__________OBTENCIÓN DE LAS MATRICES DEL SISTEMA
standardElementsL = [Elements.D1];

[K,Ml0]=computeSystemLaplaceTransient(X,T(standardElementsL,:),referenceE
lement,diffl); %K, M de los elementos estandar
[Kcut,Mcut]=computeSystemLaplaceCutElementsTransient(LS,X,T(Elements.Int,
:),referenceElement,diffl); %K, M de los elementos cortados

K=(K+Kcut); M=(Ml0+Mcut); %Suma de ambos

[K,f]=computeSystemLaplaceCutElementsNitscheTransient(Beta,K,LS,X,T(Eleme
nts.Int,:),referenceElement,diffl); %Modificación con Nitsche

Kliquid=K;
Mliquid=M;
fliquid=f;

%Nodos en el agujero (nodos que pertenecen al sólido)
nodesVoidl = find(sum(abs(K))<1.e-14)';
nodesCCDl_est = [nodesCCDl ; nodesVoidl];
uCCDl_est = [uCCDl ; zeros(size(nodesVoidl))];

%Imposición de condiciones de contorno. Reducción del sistema.
notCCDl = setdiff(1:size(X,1),nodesCCDl_est);
fliquidR = fliquid(notCCDl)-Kliquid(notCCDl,nodesCCDl_est)*uCCDl_est;
KliquidR=K(notCCDl,notCCDl);
MliquidR=M(notCCDl,notCCDl);

%CONDICIÓN INICIAL
u = zeros(size(X,1),1);
distanciaBorde = min(1-abs(X(:,1)),1-abs(X(:,2)));
aux=distanciaBorde<0.25;
u(aux)=(0.25-distanciaBorde(aux))/0.25*10;
uliquid=u;
uliquidR=u(notCCDl);

 %_____________________________________________________
%SÓLIDO

%Información del Level-set. Se obtiene cambiando el signo al del líquido
LSsolid=-LS;

%Condiciones de contorno
x = X(:,1); y = X(:,2); tol=1.e-10;
nodesCCDs = find(abs(x)<tol&abs(y)<tol); %Nodo central
figure(1), hold on, plot(x(nodesCCDs),y(nodesCCDs),'bo','MarkerSize',16);
hold off
XnodesCCDs = X(nodesCCDs,:);
uCCDs = -5*ones(size(XnodesCCDs(:,1))); %valor de contorno
```

```
%Los elementos que pertenecen al sólido son los que antes (en el líquido)
pertenecían al agujero.
standardElementsS = [Elements.D2];

%__________OBTENCIÓN DE LAS MATRICES DEL SISTEMA
[K,M0s]=computeSystemLaplaceTransient(X,T(standardElementsS,:),referenceE
lement,diffs); %K, M para elementos estandar (no cortados)
[Kcut,Mcut]=computeSystemLaplaceCutElementsTransient(LSsolid,X,T(Elements
.Int,:),referenceElement,diffs); %K, M para elementos cortados
K=(K+Kcut); M=(M0s+Mcut); %Suma
[K,f]=computeSystemLaplaceCutElementsNitscheTransient(Beta,K,LSsolid,X,T(
Elements.Int,:),referenceElement,diffs); %Modificación mediante Nitsche
Ksolid=K;
Msolid=M;
fsolid=f;

%Nodos en el agujero (el agujero corresponde a la parte del líquido)
nodesVoids = find(sum(abs(K))<1.e-14)';
nodesCCDs = [nodesCCDs ; nodesVoids];
uCCDs = [uCCDs ; zeros(size(nodesVoids))];


%CONDICIÓN INICIAL
%En la condición inicial se fija el nodo central con temperatura -5°C,
pero no es contorno, por tanto no se fija como tal.
% Imposición de condición inicial. Reducción del sistema.
notCCDs = setdiff(1:size(X,1),nodesCCDs);
fsolidR = fsolid(notCCDs)-Ksolid(notCCDs,nodesCCDs)*uCCDs;
KsolidR=K(notCCDs,notCCDs);
sols = KsolidR\fsolidR;
usolid = zeros(size(X,1),1); usolid(nodesCCDs)=uCCDs;
usolid(notCCDs)=sols;

%Condiciones de contorno. Sin fijar el nodo central.
nodesCCDs = nodesVoids;
uCCDs = zeros(size(nodesVoids));
notCCDs = setdiff(1:size(X,1),nodesCCDs);
usolidR=usolid(notCCDs);
fsolidR = fsolid(notCCDs)-Ksolid(notCCDs,nodesCCDs)*uCCDs;
KsolidR=K(notCCDs,notCCDs);
MsolidR=M(notCCDs,notCCDs);

%Plot de la solución inicial.
[Xl,Tl,ul]=XFEMtoFineLinearApproximation(LS,X,T,uliquid,10,referenceEleme
nt);
[Xs,Ts,us]=XFEMtoFineLinearApproximation(LSsolid,X,T,usolid,10,referenceE
lement);
Xtot = [Xl;Xs]; Ttot=[Tl;Ts+size(Xl,1)]; utot = [ul;us];
figure(3), clf, trisurf(Ttot,Xtot(:,1),Xtot(:,2),utot), shading interp,
view(2), axis equal, colorbar,caxis([-5,10]), colormap jet,set(gca,
'FontSize',12.5)
```

```
%___________ SOLUCION ESTACIONARIA ___________
%La solución estacionaria se obtiene resolviendo el sistema reducido.
soll = KliquidR\fliquidR;
uestl = zeros(size(X,1),1); uestl(nodesCCDl)=uCCDl; uestl(notCCDl)=soll;
sols = KsolidR\fsolidR;
uests = zeros(size(X,1),1); uests(nodesCCDs)=uCCDs; uests(notCCDs)=sols;
uliquid=uestl;
%Plot de la solución estacionaria.
[Xl,Tl,ul]=XFEMtoFineLinearApproximation(LS,X,T,uestl,10,referenceElement
);
[Xs,Ts,us]=XFEMtoFineLinearApproximation(LSsolid,X,T,uests,10,referenceEl
ement);
Xtot = [Xl;Xs]; Ttot=[Tl;Ts+size(Xl,1)]; utot = [ul;us];
figure(4), clf,trisurf(Ttot,Xtot(:,1),Xtot(:,2),utot), shading interp,
view(2), axis equal, caxis([-5,10]); colorbar, colormap jet,set(gca,
'FontSize',12.5)


%_________________SOLUCION TRANSITORIA_____________

dt=0.05; %Paso de tiempo
Ll=chol(MliquidR,'lower'); %Descomposición de Cholesky para la matriz de
masa del líquido
Ls=chol(MsolidR,'lower'); %Descomposición de Cholesky para la matriz de
masa del sólido

for pt=1:100 %Número de pasos de tiempo

    %___Método de EULER (con sistema reducido)
    uliquidR = uliquidR +dt*(Ll'\(Ll\(fliquidR-KliquidR*uliquidR)));
    usolidR = usolidR +dt*(Ls'\(Ls\(fsolidR-KsolidR*usolidR)));

    if mod(pt,10)==0 %Plot de la solución cada 10 pasos de tiempo
        uliquid(notCCDl) = uliquidR;
        usolid(notCCDs) = usolidR;

[Xl,Tl,ul]=XFEMtoFineLinearApproximation(LS,X,T,uliquid,10,referenceEleme
nt);

[Xs,Ts,us]=XFEMtoFineLinearApproximation(LSsolid,X,T,usolid,10,referenceE
lement);
        Xtot = [Xl;Xs]; Ttot=[Tl;Ts+size(Xl,1)]; utot = [ul;us];
        figure(5+pt), clf, trisurf(Ttot,Xtot(:,1),Xtot(:,2),utot),
shading interp, view(2), axis equal, colorbar, caxis([-5,10]),colormap
jet,set(gca, 'FontSize',12.5)
        title(sprintf('Paso de tiempo %d',pt))
        pause(0.1)
    end
end
```

computeSystemLaplaceTransient

```
%Este código cálcula la matriz K de rigidez y la matriz M de masa para el
problema transitorio. Es similar al código que obtiene la matriz K y el
vector f para la EDP estacionaria, simplemente se obtiene la matriz de masa
en vez del vector f modificando la integración numérica.

function [K,M]=computeSystemLaplaceTransient(X,T,referenceElement,diffl)

GaussPoints=referenceElement.IPcoordinates;
GaussWeights=referenceElement.IPweights;
N=referenceElement.N;
Nxi=referenceElement.Nxi;
Neta=referenceElement.Neta;

nOfNodes = size(X,1);
nOfElements = size(T,1);
K=spalloc(nOfNodes,nOfNodes,nOfNodes*5);
M=spalloc(nOfNodes,nOfNodes,nOfNodes*5);

%Bucle en elementos
for i=1:nOfElements
    Te=T(i,:);
    Xe=X(Te,:);

[Ke,Me]=computeElementalMatricesTransient(Xe,GaussPoints,GaussWeights,N,N
xi,Neta,diffl);
    K(Te,Te)=K(Te,Te)+Ke;
    M(Te,Te)=M(Te,Te)+Me;

end
%_____________________________________________
%Obtención de matriz y vector elementales
function
[Ke,Me]=computeElementalMatricesTransient(Xe,GaussPoints,GaussWeights,N,N
xi,Neta,diffl)
nOfNodes = size(Xe,1);
Ke=zeros(nOfNodes);
Me=zeros(nOfNodes);
fe=zeros(nOfNodes,1);
xe = Xe(:,1); ye = Xe(:,2);
%Bucle en puntos de integración
for k=1:length(GaussWeights)
    Nk=N(k,:);
    dNkdxi=Nxi(k,:);
    dNkdeta=Neta(k,:);
    xk = Nk*Xe;
    %Integración numérica
    J = [dNkdxi*xe dNkdxi*ye;dNkdeta*xe dNkdeta*ye];
    Gk = J\[dNkdxi;dNkdeta];
    dxy=GaussWeights(k)*det(J);
    Ke = Ke + diffl*Gk'*Gk*dxy;
    Me = Me + Nk'*Nk*dxy; Información de la matriz de masa M de la forma
débil

end
```

computeSystemLaplaceCutElementsTransient

```
%El objetivo de este código es obtener la matriz de masa M y la matriz de
rigidez K para el problema transitorio.
%Este código es igual que computeSystemLaplaceCutElements pero en vez de
obtener la matriz de rigidez K y el vector de términos independientes f,
se obtiene K (igual) y la matriz de masa M. Por lo tanto lo único que
cambia es la obtención M en la integración numérica.

function [K,M]=
computeSystemLaplaceCutElementsTransient(LS,X,T,referenceElement,diff)

nOfNodes = size(X,1);
nOfElements = size(T,1);

K=spalloc(nOfNodes,nOfNodes,5*nOfNodes);
M=spalloc(nOfNodes,nOfNodes,nOfNodes*5);

nDeg=referenceElement.degree;

[zgp_qua,wgp_qua] = GaussLegendreCubature2Dquad(nDeg+1);
zgp_tri = referenceElement.IPcoordinates;
wgp_tri = referenceElement.IPweights;


%Bucle en elementos
for i=1:nOfElements
    Te=T(i,:);
    Xe=X(Te,:);
    LSe=LS(Te);
    [GaussPoints,GaussWeights,n1,n2,PtsInt,CutFaces] =
ModifyQuadrature(LSe,referenceElement,zgp_tri,wgp_tri,zgp_qua,wgp_qua);
    GaussPoints=GaussPoints(1:n1,:); GaussWeights=GaussWeights(1:n1);

shapeFunctions=computeShapeFunctionsAtPoints(nDeg,referenceElement.NodesC
oord,GaussPoints);
    N = shapeFunctions(:,:,1)';
    Nxi = shapeFunctions(:,:,2)';
    Neta =shapeFunctions(:,:,3)';

[Ke,Me]=computeElementalMatricesCutElementsTransient(LSe,Xe,GaussPoints,G
aussWeights,N,Nxi,Neta,diff);
    K(Te,Te)=K(Te,Te)+Ke;
    M(Te,Te)=M(Te,Te)+Me;
end

%_____________________________________________
%Obtención de matriz y vector elementales
function
[Ke,Me]=computeElementalMatricesCutElementsTransient(LSe,Xe,GaussPoints,G
aussWeights,N,Nxi,Neta,diff)


nOfNodes = size(Xe,1);
Ke=zeros(nOfNodes);
```

```
Me=zeros(nOfNodes);
fe=zeros(nOfNodes,1);
xe = Xe(:,1); ye = Xe(:,2);

%Bucle en puntos de integración
for k=1:length(GaussWeights)
    Nk=N(k,:);
    dNkdxi=Nxi(k,:);
    dNkdeta=Neta(k,:);
    xk = Nk*Xe;

    J = [dNkdxi*xe dNkdxi*ye;dNkdeta*xe dNkdeta*ye];

    Gk = J\[dNkdxi;dNkdeta];
    %Integración numérica
    dxy=GaussWeights(k)*det(J);
    Ke = Ke + diff*Gk'*Gk*dxy;
    Me = Me + Nk'*Nk; %Información de la matriz de masa M de la forma
débil
end
```

computeSystemLaplaceCutElementsNitscheTransient

```
%El objetivo de este código es modificar la matriz K y crear el vector f
con los términos del método de Nitsche para la EDP transitoria. El vector
f lo crea ya que en el problema transitorio no hay término fuente y se
compone únicamente de los términos de Nitsche.
%Este código es igual que computeSystemLaplaceCutElementsNitsche pero en
vez de obtener la matriz de rigidez K y el vector de términos
independientes f, se obtiene K (igual) y la matriz de masa M. Por lo
tanto lo único que cambia es la obtención M en la integración numérica


function [K,f]=
computeSystemLaplaceCutElementsNitscheTransient(beta,K,LS,X,T,referenceEl
ement,diff)

nOfNodes = size(X,1);
nOfElements = size(T,1);
f=zeros(nOfNodes,1);
N1D = referenceElement.N1d;
dNds1D = referenceElement.N1dxi;
z1D = referenceElement.IPcoordinates1d;
w1D = referenceElement.IPweights1d; ngauss=length(z1D);

nDeg=referenceElement.degree;

[zgp_qua,wgp_qua] = GaussLegendreCubature2Dquad(nDeg+1);
zgp_tri = referenceElement.IPcoordinates;
wgp_tri = referenceElement.IPweights;

%Bucle en elementos
for iElem=1:nOfElements
    Te = T(iElem,:);
    Xe = X(Te,:);
    LSe=LS(Te);
    [GaussPoints,GaussWeights,n1,n2,nodesInterfaceReferenceElem,CutFaces]
= ModifyQuadrature(LSe,referenceElement,zgp_tri,wgp_tri,zgp_qua,wgp_qua);
    n=length(nodesInterfaceReferenceElem)/2;
nodesInterfaceReferenceElem=reshape(nodesInterfaceReferenceElem,2,n)';

shapeFunctions=computeShapeFunctionsAtPoints(nDeg,referenceElement.NodesC
oord,nodesInterfaceReferenceElem);
    N = shapeFunctions(:,:,1)';
    nodesInterfacePhysical= N*Xe;
    xieta_integrationPoints1D = N1D*nodesInterfaceReferenceElem;
shapeFunctions=computeShapeFunctionsAtPoints(nDeg,referenceElement.NodesC
oord,xieta_integrationPoints1D);

    N = shapeFunctions(:,:,1)'; Nxi = shapeFunctions(:,:,2)'; Neta
=shapeFunctions(:,:,3)';
    J11 = Nxi*Xe(:,1); J12 = Nxi*Xe(:,2);
    J21 = Neta*Xe(:,1); J22 = Neta*Xe(:,2);
    detJ = J11.*J22-J12.*J21;
    invJ11 = spdiags(J22./detJ,0,ngauss,ngauss);
    invJ12 = spdiags(-J12./detJ,0,ngauss,ngauss);
```

```
    invJ21 = spdiags(-J21./detJ,0,ngauss,ngauss);
    invJ22 = spdiags(J11./detJ,0,ngauss,ngauss);

    Nx = invJ11*Nxi + invJ12*Neta;
    Ny = invJ21*Nxi + invJ22*Neta;


[K2e,K3e,K4e,f2e,f3e]=computeElementalMatricesCutElementsTransient(LSe,Xe
,GaussPoints,GaussWeights,ngauss,N1D,dNds1D,N,Nx,Ny,nodesInterfacePhysica
l,w1D,diff);
    K(Te,Te)=K(Te,Te)-K2e-K3e+beta*K4e;
    f(Te) = f(Te) - f2e + beta*f3e;
end

%_____________________________________________
%Obtención de matriz y vector elementales
function
[K2e,K3e,K4e,f2e,f3e]=computeElementalMatricesCutElementsTransient(LSe,Xe
,GaussPoints,GaussWeights,ngauss,N1D,dNds1D,N,Nx,Ny,nodesInterfacePhysica
l,w1D,diff)

nOfNodes = size(Xe,1);
K2e=zeros(nOfNodes);
K3e=zeros(nOfNodes);
K4e=zeros(nOfNodes);
f2e=zeros(nOfNodes,1);
f3e=zeros(nOfNodes,1);
xe = Xe(:,1); ye = Xe(:,2);
    for g = 1:ngauss
        N1D_g = N1D(g,:);
        N2D_g = N(g,:); N2D_dx_g = Nx(g,:); N2D_dy_g = Ny(g,:);
        G_g = [N2D_dx_g;N2D_dy_g];
        dxds_g = dNds1D(g,:);
        xg = N1D_g*nodesInterfacePhysical
        dxds_g = dxds_g*nodesInterfacePhysical;
        norma = norm(dxds_g);
        dl = w1D(g)*norma;
        t = dxds_g/norma;
        n = [t(2),-t(1)];
        K2e = K2e + diff*N2D_g'*(n*G_g)*dl;
        K3e = K3e + diff*(G_g'*n')*N2D_g*dl;
        K4e = K4e + diff*(N2D_g'*N2D_g)*dl;
        f2e = f2e + diff*zeros(size(xg(:,1)))*(G_g'*n')*dl;
        f3e = f3e + diff*zeros(size(xg(:,1)))*N2D_g'*dl;
    end
```

mainTransientMovingInterface

```
%Éste es el código principal que soluciona la EDP transitoria considerando
una interfaz móvil.
%La diferencia con el código "mainTransientNotMovingInterface" es que el
dominio de cada uno de los materiales cambia en cada paso de tiempo. Habrá
que calcular la velocidad normal, realizar una proyección de la solución y
calcular las matrices K, M y el vector f en cada paso de tiempo.

clear all, clc, close all
setpath

global centre radius
centre=[0,0]; radius=0.5; %Centro y radio de la parte sólida.

%_________________PREPROCESO
%Malla (X, T) sobre el dominio y grado de interpolación
degree = 1;
numMallas = 3;
meshNum=numMallas;
meshName = sprintf('mesh%d_P%d.cdm',meshNum,degree

if all(meshName(end-2:end)=='dcm') GenerateMatFileFromEZ4U(['Meshes/'
meshName]); end
load(['Meshes/' meshName(1:end-3) 'mat']);
X = 2*X-1;
figure(1),clf
plotMesh(X,T)

%Elemento de referencia
referenceElement = createReferenceElement(1,elemInfo.nOfNodes);
Beta = 1000*(2^meshNum);

%Propiedades de los materiales
kl=6.9e-3; %Conductividad térmica del líquido
cl=0.62; %Capacidad calorífica volumétrica del líquido
ks=9.6e-3; %Conductividad térmica del sólido
cs=0.49; %Capacidad calorífica volumétrica del sólido
diffl=kl/cl;
diffs=ks/cs;
L=19.2; %calor latente de fusión



%_____________________________________________________________________
%LIQUID

% Información del Level-set
LS = EvaluateLS(X);
t=linspace(0,2*pi,101); figure(1), hold on,
plot(radius*cos(t),radius*sin(t),'LineWidth',2), hold off
Elements = SetElements(T,LS,[1,0],referenceElement);
%Nodos en el contorno
x = X(:,1); y = X(:,2); tol=1.e-10;
nodesCCDl = find(abs(x+1)<tol|abs(x-1)<tol|abs(y+1)<tol|abs(y-1)<tol);
```

```
figure(1), hold on, plot(x(nodesCCDl),y(nodesCCDl),'bo','MarkerSize',16);
hold off
XnodesCCDl = X(nodesCCDl,:);  %coordenadas de los nodos del contorno
uCCDl = 10*ones(size(XnodesCCDl(:,1))); %valor del contorno


%__________OBTENCIÓN DE LAS MATRICES DEL SISTEMA
standardElementsL = [Elements.D1];
[K,Ml0]=computeSystemLaplaceTransient(X,T(standardElementsL,:),referenceE
lement,diffl); %K, M de los elementos estandar
[Kcut,Mcut]=computeSystemLaplaceCutElementsTransient(LS,X,T(Elements.Int,
:),referenceElement,diffl); %K, M de los elementos cortados
K=(K+Kcut); M=(Ml0+Mcut); %Suma de ambos
[K,f]=computeSystemLaplaceCutElementsNitscheTransient(Beta,K,LS,X,T(Eleme
nts.Int,:),referenceElement,diffl); %Modificación con Nitsche
Kliquid=K;
Mliquid=M;
fliquid=f;

%Nodos en el agujero (nodos que pertenecen al sólido)
nodesVoidl = find(sum(abs(K))<1.e-14)';
nodesCCDl_est = [nodesCCDl ; nodesVoidl];
uCCDl_est = [uCCDl ; zeros(size(nodesVoidl))];


%Imposición de condiciones de contorno. Reducción del sistema.
notCCDl = setdiff(1:size(X,1),nodesCCDl_est);
fliquidR = fliquid(notCCDl)-Kliquid(notCCDl,nodesCCDl_est)*uCCDl_est;
KliquidR=K(notCCDl,notCCDl);
MliquidR=M(notCCDl,notCCDl);

%CONDICIÓN INICIAL
u = zeros(size(X,1),1);
distanciaBorde = min(1-abs(X(:,1)),1-abs(X(:,2)));
aux=distanciaBorde<0.25;
u(aux)=(0.25-distanciaBorde(aux))/0.25*10;
uliquid=u;
uliquidR=u(notCCDl);



%_____________________________________________________
%SÓLIDO

%Información del Level-set. Se obtiene cambiando el signo al del líquido
LSsolid=-LS;
x = X(:,1); y = X(:,2); tol=1.e-10;
nodesCCDs = find(abs(x)<tol&abs(y)<tol); %Nodo central
figure(1), hold on, plot(x(nodesCCDs),y(nodesCCDs),'bo','MarkerSize',16);
hold off
XnodesCCDs = X(nodesCCDs,:);
uCCDs = -5*ones(size(XnodesCCDs(:,1))); %valor de contorno
```

```
%Los elementos que pertenecen al sólido son los que antes (en el líquido)
pertenecían al agujero.
standardElementsS = [Elements.D2];

%__________OBTENCIÓN DE LAS MATRICES DEL SISTEMA
[K,M0s]=computeSystemLaplaceTransient(X,T(standardElementsS,:),referenceE
lement,diffs); % K, M para elementos estandar (no cortados)
[Kcut,Mcut]=computeSystemLaplaceCutElementsTransient(LSsolid,X,T(Elements
.Int,:),referenceElement,diffs); % K, M para elementos cortados
K=(K+Kcut); M=(M0s+Mcut); %Suma
[K,f]=computeSystemLaplaceCutElementsNitscheTransient(Beta,K,LSsolid,X,T(
Elements.Int,:),referenceElement,diffs); %Modificación mediante Nitsche
Ksolid=K;
Msolid=M;
fsolid=f;

%Nodos en el agujero (el agujero corresponde a la parte del líquido)
nodesVoids = find(sum(abs(K))<1.e-14)';
nodesCCDs = [nodesCCDs ; nodesVoids];
uCCDs = [uCCDs ; zeros(size(nodesVoids))];


%CONDICIÓN INICIAL
%En la condición inicial se fija el nodo central con temperatura -5°C,
pero no es contorno, por tanto no se fija como tal.
% Imposición de condición inicial. Reducción del sistema.
notCCDs = setdiff(1:size(X,1),nodesCCDs);
fsolidR = fsolid(notCCDs)-Ksolid(notCCDs,nodesCCDs)*uCCDs;
KsolidR=K(notCCDs,notCCDs);
sols = KsolidR\fsolidR;
usolid = zeros(size(X,1),1); usolid(nodesCCDs)=uCCDs;
usolid(notCCDs)=sols;

%Condiciones de contorno. Sin fijar el nodo central.
nodesCCDs = nodesVoids;
uCCDs = zeros(size(nodesVoids));
notCCDs = setdiff(1:size(X,1),nodesCCDs);
fsolidR = fsolid(notCCDs)-Ksolid(notCCDs,nodesCCDs)*uCCDs;
KsolidR=K(notCCDs,notCCDs);
MsolidR=M(notCCDs,notCCDs);
usolidR=usolid(notCCDs);


%Plot de la solución inicial.
[Xl,Tl,ul]=XFEMtoFineLinearApproximation(LS,X,T,uliquid,10,referenceEleme
nt);
[Xs,Ts,us]=XFEMtoFineLinearApproximation(LSsolid,X,T,usolid,10,referenceE
lement);
Xtot = [Xl;Xs]; Ttot=[Tl;Ts+size(Xl,1)]; utot = [ul;us];
figure(3), clf, trisurf(Ttot,Xtot(:,1),Xtot(:,2),utot), shading interp,
view(2), axis equal, colorbar,caxis([-5,10]), colormap jet,set(gca,
'FontSize',12.5)
```

```
%_________ SOLUCION ESTACIONARIA ___________
%La solución estacionaria se obtiene resolviendo el sistema reducido.
soll = KliquidR\fliquidR;
uestl = zeros(size(X,1),1); uestl(nodesCCDl)=uCCDl; uestl(notCCDl)=soll;
sols = KsolidR\fsolidR;
uests = zeros(size(X,1),1); uests(nodesCCDs)=uCCDs; uests(notCCDs)=sols;
uliquid=uestl;
%Plot de la solución estacionaria.
[Xl,Tl,ul]=XFEMtoFineLinearApproximation(LS,X,T,uestl,10,referenceElement
);
[Xs,Ts,us]=XFEMtoFineLinearApproximation(LSsolid,X,T,uests,10,referenceEl
ement);
Xtot = [Xl;Xs]; Ttot=[Tl;Ts+size(Xl,1)]; utot = [ul;us];
figure(4), clf,trisurf(Ttot,Xtot(:,1),Xtot(:,2),utot), shading interp,
view(2), axis equal, caxis([-5,10]); colorbar, colormap jet,set(gca,
'FontSize',12.5)

%_________________SOLUCION TRANSITORIA_____________

dt=0.07; %Paso de tiempo
LS0=LS; %Level-Set inicial

for pt=1:20 %Número de pasos de tiempo

    %_____________Cálculo del Nuevo level-set
      %Velocidad normal
    numpuntos=10; %Número de puntos en la interfaz en un mismo elemento.
      [vns,puntos,nvns]=calculaVelocidadNormalInterfaz(numpuntos,uliquid,
      usolid,kl,ks,referenceElement,LS0,X,T(Elements.Int,:));
    vns = vns/L;
          figure(9),plotMesh(X,T), hold on,
      quiver(puntos(:,1),puntos(:,2),nvns(:,1),nvns(:,2),'r'), set(gca,
      'FontSize',12.5),hold off,
    t=linspace(0,2*pi,101); figure(9), hold on,
    plot(radius*cos(t),radius*sin(t),'b','LineWidth',2), hold off
      %Velocidad específica
    figure(5),
    for i=1:length(X)
        ptsaux=puntos;
        [k,d] = dsearchn(ptsaux,X(i,:));
        ve(i)=vns(k);
    end
    hold on
    for j=1:numpuntos:length(puntos)-1
        figure(5), plot(puntos(j:j+numpuntos-1,1),puntos(j:j+numpuntos-
    1,2),'r-'),set(gca, 'FontSize',12.5)
    end
    hold off
    LSnew=LS0+ve'*dt;
    LSsolidnew=-LSnew;
    %_____________Proyección L2
    Elements0 = SetElements(T,LS0,[1,0],referenceElement);
    ElementsNew = SetElements(T,LSnew,[1,0],referenceElement);
    umelting=0;
    cutElements = union(Elements0.Int,ElementsNew.Int);
```

```
  uliquid=computeL2projection2levelset(umelting,uliquid,cutElements,L
  S0,X,T,LSnew,Ml0,referenceElement);


%_____________MATRICES DEL LÍQUIDO
standardElementsL = [Elements0.D1];

  [Kl,Ml0]=computeSystemLaplaceTransient(X,T(standardElementsL,:),ref
  erenceElement,diffl);
  [Kcut,Mcut]=computeSystemLaplaceCutElementsTransient(LS,X,T(Element
  s0.Int,:),referenceElement,diffl);
Kliquid=(Kl+Kcut);

  [Kliquid,fliquid]=computeSystemLaplaceCutElementsNitscheTransient(B
  eta,Kliquid,LS,X,T(Elements0.Int,:),referenceElement,diffl);
Mliquid = Ml0 + Mcut;

  % Imposición de condición inicial. Reducción del sistema.
nodesVoidl_trans = find(sum(abs(Mliquid))<1.e-10*max(abs(diag(M))))';
nodesCCDl_trans = [nodesCCDl ; nodesVoidl_trans];
notCCDl_trans = setdiff(1:size(X,1),nodesCCDl_trans);
uCCDl_trans = [uCCDl ; zeros(size(nodesVoidl_trans))];
fliquidR = fliquid(notCCDl_trans)-
Kliquid(notCCDl_trans,nodesCCDl_trans)*uCCDl_trans;
KliquidR=Kliquid(notCCDl_trans,notCCDl_trans);
MliquidR=Mliquid(notCCDl_trans,notCCDl_trans);
uliquidR=uliquid(notCCDl_trans);
uliquid(nodesCCDl_trans) = uCCDl_trans;


%_____________ MATRICES DEL SÓLIDO
standardElementsS = [Elements0.D2];

  [Ks,Ms]=computeSystemLaplaceTransient(X,T(standardElementsS,:),refe
  renceElement,diffs);

  [Kcut,Mcut]=computeSystemLaplaceCutElementsTransient(LSsolid,X,T(El
  ements0.Int,:),referenceElement,diffs);
Ksolid=(Ks+Kcut); Msolid=(Ms+Mcut);

  [Ksolid,fsolid]=computeSystemLaplaceCutElementsNitscheTransient(Bet
  a,Ksolid,LSsolid,X,T(Elements0.Int,:),referenceElement,diffs);

  % Imposición de condición inicial. Reducción del sistema.
nodesVoids_trans = find(sum(abs(Ksolid))<1.e-14)';
nodesCCDs_trans = [nodesVoids_trans];
notCCDs_trans = setdiff(1:size(X,1),nodesCCDs_trans);
uCCDs_trans = [zeros(size(nodesVoids_trans))];
fsolidR = fsolid(notCCDs_trans)-
Ksolid(notCCDs_trans,nodesCCDs_trans)*uCCDs_trans;
KsolidR=Ksolid(notCCDs_trans,notCCDs_trans);
MsolidR=Msolid(notCCDs_trans,notCCDs_trans);
usolidR=usolid(notCCDs_trans);
usolid(nodesCCDs_trans)=uCCDs_trans;
```

```
    %_____________EULER_____________________
    Ll=chol(MliquidR,'lower');
    Ls=chol(MsolidR,'lower');
    uliquidR = uliquidR +dt*(Ll'\(Ll\(fliquidR-KliquidR*uliquidR)));
    usolidR = usolidR +dt*(Ls'\(Ls\(fsolidR-KsolidR*usolidR)));



    uliquid(notCCDl_trans) = uliquidR;
    usolid(notCCDs_trans) = usolidR;


    if mod(pt,5)==0

        %Plot solución cada 5 pasos
      [Xl,Tl,ul]=XFEMtoFineLinearApproximation(LSnew,X,T,uliquid,1,refere
      nceElement);

      [Xs,Ts,us]=XFEMtoFineLinearApproximation(LSsolidnew,X,T,usolid,1,re
      ferenceElement);
        Xtot = [Xl;Xs]; Ttot=[Tl;Ts+size(Xl,1)]; utot = [ul;us];
        figure(6), clf, trisurf(Ttot,Xtot(:,1),Xtot(:,2),utot), shading
interp, view(2), axis equal, colorbar, caxis([-5,10]), colormap
jet,set(gca, 'FontSize',12.5)
        title(sprintf('Paso de tiempo pt=%d',pt))
        pause(0.1)
    end
        LS0=LSnew;
end
```

calculaVelocidadNormalInterfaz

```
%Esta función calcula la velocidad normal en numerosos puntos de la
interfaz. Como inputs tiene: el número de puntos en cada elemento en los
que se desea calcular la velocidad, la solución del líquido y la del
sólido, las conductividades térmicas de ambos materiales, el elemento de
referencia, el level-set y la malla (X,T). Como outpus tiene las
velocidades (vns), los puntos en los que didchas velocidades están
calculadas (puntos) y las velocidades multiplicadas por la normal del
líquido (nvns).

function [vns,puntos,nvns]=
calculaVelocidadNormalInterfaz(numpuntos,uliquid,usolid,kl,ks,referenceEl
ement,LS,X,T)
%La matriz de conectividad T sólo contiene información de los elementos
cortados, que son los que están en la interfaz.

N1D = referenceElement.N1d; %Funciones de forma del element de referencia
dNds1D = referenceElement.N1dxi; %Derivadas de las funciones de forma

Points1D=linspace(-1,1,numpuntos); npoints=length(Points1D); %Puntos
sobre el elemnto de referencia 1D en los que se calculará la velocidad

nOfElements = size(T,1); %Número de elementos cortados
nDeg=referenceElement.degree; %Grado de interpolación

%puntos y pesos del cuádrilatero y del triángulo de ref. Necesarios para
cálculos intermedios.
[zgp_qua,wgp_qua] = GaussLegendreCubature2Dquad(nDeg+1);
zgp_tri = referenceElement.IPcoordinates;
wgp_tri = referenceElement.IPweights;


vns = []; puntos = []; nvns=[]; %Inicialización de las variables

%Bucle en elementos
for iElem = 1:nOfElements
    %Información del elemento i
    Te = T(iElem,:);
    Xe = X(Te,:);
    LSe=LS(Te);
    uliquide=uliquid(Te,:);
    usolide=usolid(Te,:);

%Con la función "modifyQuadrature" se obtienen los nodos de la interfaz
en el elemento de referencia
    [GaussPoints,GaussWeights,n1,n2,nodesInterfaceReferenceElem,CutFaces]
= ModifyQuadrature(LSe,referenceElement,zgp_tri,wgp_tri,zgp_qua,wgp_qua);
    n=length(nodesInterfaceReferenceElem)/2;
nodesInterfaceReferenceElem=reshape(nodesInterfaceReferenceElem,2,n)';

%Se obtienen el valor de las funciones de forma en dichos nodos
shapeFunctions=computeShapeFunctionsAtPoints(nDeg,referenceElement.NodesC
oord,nodesInterfaceReferenceElem);
    N = shapeFunctions(:,:,1)';
```

```
%Con las funciones de forma, se pueden obtener los nodos en el elemento
físico
    nodesInterfacePhysical= N*Xe;

 %Valor de las funciones de forma 1D y derivadas en los puntos en los que
se calculará la velocidad
shapeFunctionsPoints1D=computeShapeFunctionsAtPoints(nDeg,referenceElemen
t.NodesCoord1d,Points1D);
    NPoints1D = shapeFunctionsPoints1D(:,:,1)';
    N_s_Points1D=shapeFunctionsPoints1D(:,:,2)'
%Puntos en los que se calculará la velocidad en el elemento de
referencia. Se obtienen multiplicando el valor de las funciones de forma
en dichos puntos por las coordenadas de los nodos de la interfaz en el
elemento de referencia.
    PointsReferenceElement = NPoints1D*nodesInterfaceReferenceElem;

%Valor de las funciones de forma 2D en los puntos en los que se calculará
la velocidad en el elemento de referencia

shapeFunctionsPoints2D=computeShapeFunctionsAtPoints(nDeg,referenceElemen
t.NodesCoord,PointsReferenceElement);
    NPoints2D = shapeFunctionsPoints2D(:,:,1)';

%Puntos en los que se calculará la velocidad en el elemento físico.
Multiplicas el valor de las funciones de forma en dichos puntos sobre el
elemento de referencia multiplicado por Xe, las coordenadas de los nodos
del elemento físico.
    PointsPhysical = NPoints2D*Xe;

%Funciones de forma y derivadas en puntos sobre los que se calculará la
velocidad normal en el elemento de referencia y cálculos para conseguir
las derivadas de esas funciones con respecto a los ejes 'x' e 'y'
    N = shapeFunctions(:,:,1)'; Nxi = shapeFunctionsPoints2D(:,:,2)';
    Neta =shapeFunctionsPoints2D(:,:,3)';
    J11 = Nxi*Xe(:,1); J12 = Nxi*Xe(:,2);
    J21 = Neta*Xe(:,1); J22 = Neta*Xe(:,2);
    detJ = J11.*J22-J12.*J21;
    ngauss=npoints;
    invJ11 = spdiags(J22./detJ,0,ngauss,ngauss);
    invJ12 = spdiags(-J12./detJ,0,ngauss,ngauss);
    invJ21 = spdiags(-J21./detJ,0,ngauss,ngauss);
    invJ22 = spdiags(J11./detJ,0,ngauss,ngauss);
    %Derivadas con respecto a 'x' e 'y'
    Nx = invJ11*Nxi + invJ12*Neta;
    Ny = invJ21*Nxi + invJ22*Neta;

    %Bucle en puntos sobre los que se calculará la velocidad
    for g = 1:npoints
        %Valor de funciones de forma
        NPoints1D_g = NPoints1D(g,:);
        N2D_g = NPoints2D(g,:); N2D_dx_g = Nx(g,:); N2D_dy_g = Ny(g,:);
        %Matriz de derivadas
        G_g = [N2D_dx_g;N2D_dy_g];
        Grad_ul = G_g*uliquide; %Gradiente de ulíquido
        Grad_us = G_g*usolide; %Gradiente de usólido
```

```
        %Integración numérica
        dxds_g = N_s_Points1D(g,:);
        xg = NPoints1D_g*nodesInterfacePhysical; %coordenadas de los
        puntos de integración en el element físico (x,y)
        dxds_g = dxds_g*nodesInterfacePhysical;
        norma = norm(dxds_g);
        t = dxds_g/norma;
        n = [t(2),-t(1)];
        vn = -n*(kl*Grad_ul-ks*Grad_us);%Velocidad normal.
        puntos = [puntos;xg]; vns=[vns,vn]; nvns=[nvns; vn*n];
    end

end
```